Home Smart Home: Approachable Interfaces for Intelligibility, Modification, and End-User Programming.

Michaela R. Reisinger[1], Johann Schrammel[1], Stefan Suette[1], Peter Fröhlich [1]

[1] AIT Austrian Institute of Technology, Giefinggasse 4, 1210 Vienna
{firstname.lastname}@ait.ac.at

Abstract. End-user programming concepts are increasingly employed in smart home research to address the growing complexity of controlling smart home environments. Different approaches and visual styles of end-user programming have been proposed and implemented within this context. Smart home control does however not only necessitate end-user programming but also understanding and modifying existing program structures. In this study, we compare three different approaches regarding their suitability for this application context with a specific focus on intelligibility and modification performance. We conducted an empirical study with 39 users performing three types of tasks (understanding, configuring, and programming), using three different approaches for end-user programming (form-filling, data-flow, and grid-canvas). The results of our study found no significant differences regarding the intelligibility of the three different implementations but clear differences in the subjective preference of users as well as configuration and programming performance.

Keywords: End-User Programming, End-User Development, Smart Home, Home Automation, Smart Environments, Rule-Based Processing, Trigger-Action Programming.

1 Introduction & Related Work

Smart homes execute control over a multitude of devices. While this is only one of several dimensions of a smart home [1, 2], creating approachable configuration interfaces is a central goal in smart home research [3], since it is residents rather than professional programmers who are most suited to this task [4]. Programming can be seen as an extension of direct manipulation, facilitating repeated actions and scheduling [5]. As such, programming (i.e. creating an abstraction of a task), must have a decidedly better cost-benefit ratio compared to manual control [5]. This abstractive effort can be eased using visual programming languages and employing metaphors, even though they impact the language's expressiveness, i.e. the number of relevant expressible concepts, [6, 7], and its complexity, i.e. the number of available items [8].

1.1 End-User Programming Approaches for a Home Context

To reach the goal of an approachable interface, combinations of different techniques including metaphors and programming styles have been proposed. While metaphors employ real-world concepts without specific programming relevance (e.g. jigsaw, timelines, pipes, or rules), programming styles make use of specific programming interaction paradigms (e.g. trigger-action, natural language, tangible, spreadsheet) [9].
Since end-user programming interfaces can use these metaphors and programming styles in different ways, we additionally describe them by four principles: their *structure*, which can be pre-defined, like a form, or open, like an empty canvas, their *developmental process*, in which starting point and/or progression through the programming steps is either open or fixed (determined either by the user or the system respectively), the *developmental logic*, which can make explicit or implicit use of logic markers such as if/then, and the item *repository*, which can be general (i.e. unchanging throughout the programming experience) or position-based (i.e. different subsets of the repository are shown at different stages of development). Repositories can furthermore use categories or user-input/search for browsing.

Investigating the home context and available end-user programming interfaces show a prevalence of certain techniques employed in smart homes: Household tasks are commonly expressed as rules, following an if-then structure [3, 8, 10], therefore the trigger-action paradigm is especially widespread in the home context [e.g. 11–14]. Yet, rule-notions have been noted as difficult for non-programmers, especially with rising complexity, because they necessitate the use of key concepts like Boolean operators and operator priority [9]. Rule-based interfaces have also been shown to limit the user's creativity [3]: Users described feeling restricted by it and were less expressive using it. Therefore, other approaches should continue to be considered, particularly their potential to amend the shortcomings of rule-based interfaces. Process-oriented notions engage users to model more complex tasks than rule-based notions and facilitate expressiveness, but on the other hand lack clarity. Comparing existing approaches, repository categorization, used terminology, developmental process, complexity and a low cut-off for expressiveness have been shown to affect performance and perception [4, 6, 15–17]. To more clearly contrast the effect of metaphors or programming styles themselves, controlling aspects that are merely properties of their implementation (and do not define the underlying approach) such as styling, terminology, complexity and expressiveness seems warranted.

Previous research has already sought to address common issues, such as intelligibility and appeal for end-users by guiding users through the programming experience by separating rule parts [11], visualizing differences between trigger-action programs [18] and recommending rules through algorithms [19]. Since these approaches have not yet been tested with the anticipated complexity of a real-life home, it is still open whether they address the issues most difficult for end-users. Therefore, programming for smart homes has still to be investigated at higher complexity levels and different approaches tested to this point. Since previous studies have also shown that programming can make use of one metaphor while debugging can successfully make use of another [20], different approaches might be used according to the task at hand. In the following, we therefore outline code modification and readability as important tasks that complement the classical task of programming.

1.2 Code modification and readability

In the context of end-user programming for smart home control, the modification of existing ‘code’ is a very common situation that should be supported accordingly. Modifying existing behaviors is part of one of six design principles for smart home control [17]. Furthermore, code readability has also not yet received much attention concerning the impact of different approaches to end-user programming, though it has been identified as an important aspect of end-user programming [21, 22]. Code readability is a pre-requisite for successful code modification as well as a common theme in programming research [e.g. 23–25]. Understandability and readability of code are especially important in our targeted context due to the following three aspects: First, smart home controls are typically installed with default programs and configurations [26, 27]. To be able to use the system successfully and implement modifications the end-user programmer needs to be able to understand the pre-existing code structure. Second, smart home controls typically are used by multiple users (i.e. different members of the household) [28] and therefore understanding code others have changed last is typical. Third, changes also might be made only very seldom and after longer times as practices in the home are relatively stable. Typical triggers might be holidays or the start of the heating season, and people will have to understand code they made some time ago. Therefore, comparing end-user programming approaches should include code modification and intelligibility tasks just as much as programming tasks.

2 Research Questions

With our work, we compare three different approaches to end-user programming, form-filling, data-flow and grid-canvas (see section 3.2), regarding their usefulness and understandability for end-users. Specifically, we address the following research questions: (1) Are there differences in the intelligibility of pre-existing end-user programs for three different interfaces, and if, what are the underlying reasons and elements causing these differences? (2) Which interface style allows end-users to perform the best regarding both programming/modification time and error rate, and how can the observed difference be explained? (3) Are there any typical errors associated with a specific interface style, and how can these be avoided? (4) What is the subjective satisfaction of end-user-programmers with the different interface approaches, and how could the different approaches be improved to better support them and increase satisfaction?

3 Method

3.1 Participants

20 women and 19 men participated in the study. They were between 22 and 71 years of age (M = 41.15). They indicated basic to extensive computer knowledge (M = 5.02 on

a scale from 1 basic to 7 extensive) and were regular computer users (34 daily, 5 multiple times a week). Only few individuals were regularly concerned with programming: 24 did never do any programming at all, while ten, two and four individuals programmed multiple times a year, month, and week respectively, and one daily. 31 participants had never used smart home functionalities while as smaller number did so regularly (1, 1, 3, 3 multiple times per year, month, week, and daily respectively). Consequently, participants reported their smart home knowledge from non-existent to advanced (M = 3, on a scale from 1 none to 7 extensive).

3.2 Prototypes

This study used three prototypes (Fig. 1) to configure and program smart home rules by defining triggers and subsequent actions. According to the classification in [9], all prototypes employ a rule-metaphor while using different visualizations. Two of the used prototypes, form-filling and data-flow, are based on a previous study [4]. We employed prototypes instead of available tools in order to control for aspects highlighted in previous comparisons (see section 1.1): All three used the same vocabulary, the same repository size, the same color-coding and were of the same complexity (70 items each). The cut-off for expressivity was high in all prototypes and was not reached by any participant.

Form-filling is a prototype with a pre-defined structure, fixed starting point, open order, and explicit logic markers. Any drop-down choice is automatically followed by necessary specification fields. Due to its position-based repository, some items are only available either in trigger or action drop-downs (e.g. someone) and as a specification to appropriate objects (e.g. the alarm system can be active while the window cannot). Form-filling is closest to natural language use in defining rules.

Data-flow represents a more graphical, tangible approach by featuring an open canvas with a general, categorized repository. Categories are locations, objects, conditions, actions, individuals, events, time and logic. Data-flow uses explicit logic markers, in contrast to its first version [4]. The developmental process is open.

Grid-canvas is a prototype containing a segmented canvas and a general, categorized repository using the same categories as data-flow (except "logic", since these objects were embedded in the canvas). The canvas is parted in two sides (if/then) which hold containers for item and specification placement. It has explicit logic markers. Boolean operators are chosen via drop-down between containers and automatically appear when a new container is created. Like the other prototypes, it features an open starting point and an open order. Regarding visual programming representations, it could be classed as a visual, drag-and-drop variant of form-filling. The motivation for this prototype is based on a previous study [4, 16], which showed that designated if/then layers and embedding of operators could improve an open canvas approach, especially regarding the connection completion rates.

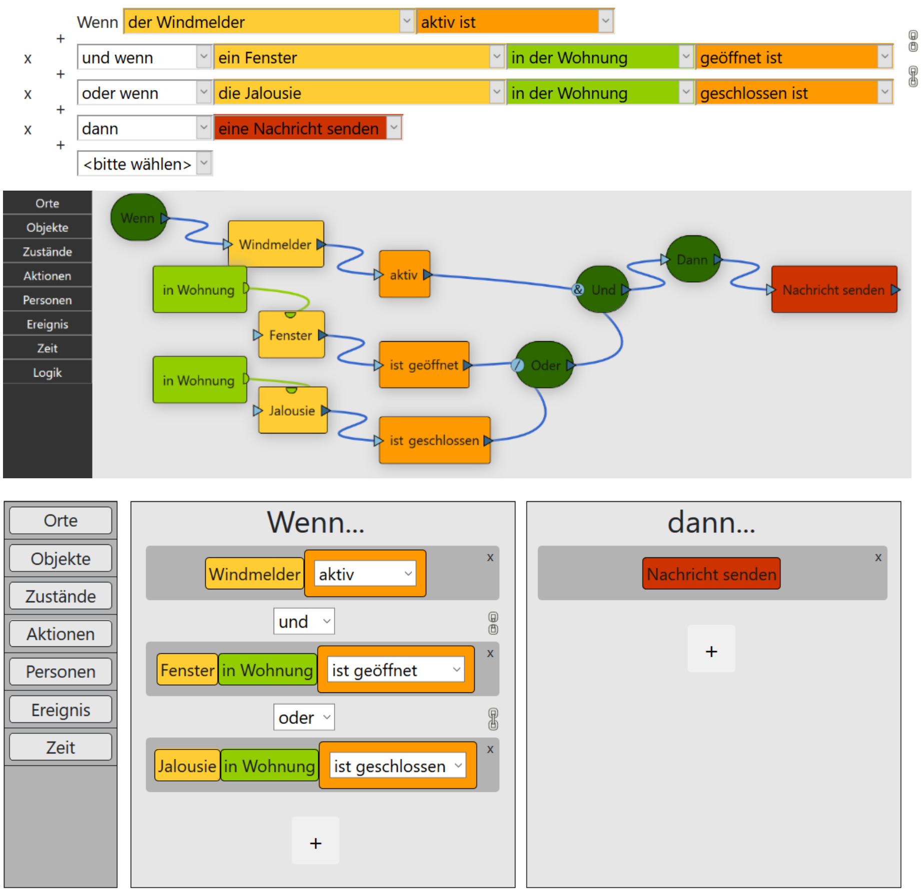

Fig. 1. An example of a rule (intelligibility task 2) in all three prototypes. It reads "If | [the] wind sensor | [is] active, | and [if either] | [a] window | in the flat | is open, | or [if] | [any] blind | in the flat | is closed | then | send [a] message". **Form-filling (top).** Symbols of the left add/remove whole lines, while symbols right to the rule connect if-pairs to define line grouping (open/closed chain) and thus trigger hierarchy. Every rule starts with if/when (*Wenn*), followed by a trigger-item dropdown. Subsequent lines start with a Boolean operator. Drop-down menus are shown, and their content adjusted according to the selection. In this program, the first trigger condition is made up by a trigger and its specification, while the subsequent two are followed by two specifications. The action drop-down in this rule does not have a specification field. **Data-flow (center).** The repository (left) holds categories of items which expand on click. Items can be dragged and dropped freely on the canvas. Connections are only possible between correct connectors (e.g. arrow-tip to arrow-bottom) and can be dissolved with left-click. Rules start with an if/when which can be connected to any trigger. Locations are connected to items via their top-notch, while other specifications (e.g. active) are added in the data flow, before Boolean operators. Trigger hierarchies are represented by the order in which the operators are connected. **Grid-canvas (bottom).** Items are dragged and dropped from a repository (left) into containers on either side (if/then) of the canvas. Containers can be added with the bottom "+", which also adds logic connectors between them (dropdown and, for triggers, chain-link). Each container can only hold one item and specifications available for that item.

3.3 Tasks

Participants performed two intelligibility tasks, eight configuration tasks, and three programming tasks per prototype. Three task sets were used, recreating the same structure with different items of the same item types (e.g. using *window* instead of *door*, *kitchen* instead of *bedroom*). Each task set was constructed to use items equal to other sets, and whenever possible to choose items with the same number of specification slots. All tasks can be found in Appendix A, while their structure is detailed below. Each set was equally used with each prototype.

Intelligibility Tasks. As intelligibility tasks, participants were shown a finished rule with six verbal statements (randomized order, Table 1) for each task. The statements verbalized when the rule would be active and lead to the presented action. Participants had to indicate which of the descriptions matched the rule (multiple choice). The intelligibility tasks structurally matched the target configurations of configuration tasks 3 and 7, and programming tasks 2 and 3 respectively.

Configuration Tasks. Participants were given an initial configuration and a description of the target configuration in eight configuration tasks (Table 2). The task order was set from the least complex (C 1) to most complex (C 8).

Programming Tasks. In these tasks (Table 3), participants were instructed to create programs from scratch in response to the target situation. The setup here was the same as for the configuration tasks but without any initial configuration present.

Table 1. Intelligibility tasks. For the actual tasks, see Appendix A, Tables 1 and 2.

Task	Rule Configuration [a]	Correct Answer Structures	Incorrect Answer Structures
		T_1 & $T_2 \rightarrow A$	$T_1 \rightarrow A$
I 1	T_1 & $T_2 \rightarrow A$	T_2 & $T_1 \rightarrow A$	$T_1 \mid T_2 \rightarrow A$
		$A \leftarrow T_1$ & T_2	T_1 & $T_2 \leftarrow A$
		T_1 & $T_2 \rightarrow A$	$T_2 \mid T_3 \rightarrow A$
I 2	T_1 & $(T_2 \mid T_3) \rightarrow A$	T_3 & $T_1 \rightarrow A$	T_2 & $T_3 \rightarrow A$
		$A \leftarrow T_1$ & $(T_2 \mid T_3)$	$A \leftarrow (T_1$ & $T_2) \mid T_3$

[a]. T indicates a trigger, A an action, & a conjunction, | a disjunction

Table 2. Configuration tasks. For the actual tasks, see Appendix A, Table 3.

Task	Initial Configuration [a]	Target Configuration [a]	Main task
C 1	T → A	T → A	Substitute trigger
C 2	T & T → A	T → A	Remove trigger
C 3	T → A	T & T → A	Add conjunction and trigger
C 4	T → A	T | T → A	Add disjunction and trigger
C 5	T → A	T | T → A & A	Add disjunction, trigger, and action
C 6	T & (T & T) → A	T | (T & T) → A	Change conjunction to disjunction
C 7	(T & T) | T → A	T & (T | T) → A	Change trigger hierarchy
C 8	T | T → A & A	T | T → A & A → A	Add action with time delay

[a]. T indicates a trigger, A an action, & a conjunction, | a disjunction

Table 3. Programming tasks & task matching. For the actual tasks, see Appendix A, Table 4.

Task	Target Rule [a]	Matching Intelligibility Tasks	Matching Configuration Targets
P 1	$T \rightarrow A$		C 1 / C 2
P 2	$T \& T \rightarrow A$	I 1	C 3
P 3	$T \& (T \mid T) \rightarrow A$	I 2	C 7

[a]. T indicates a trigger, A an action, & a conjunction, | a disjunction

Procedure and Measures

The order of the prototypes and task sets alternated between participants. They were given a brief explanation of the smart home context and programming for the home, then immediately started their first prototype session with the intelligibility task. We deliberately set the intelligibility tasks before the tutorial to gain insight into a prototype's intelligibility before exposure. They were then introduced to the prototype functions with a tutorial, which they could review at any time during the session. They proceeded with all eight configuration and three programming tasks. After each task, they rated the difficulty of understanding the task description (Perceived Task Comprehension Difficulty), the difficulty of configuring/programming it with the prototype (Perceived Configuration/Programming Difficulty) and their perceived success of doing so (Perceived Success) on a 7-point scale. After the last programming task, participants answered the User Experience Questionnaire (UEQ, [29]) for the current prototype and started with the next. The prototypes recoded the time from starting a task until participants indicated their program to be completed (Task completion time).

An Intelligibility Score was calculated by subtracting false from correct choices (cut-off at 0) and dividing by the number of total correct answers. Configuration and Programming were scored by analyzing participants' final programs according to the use of correct items (e.g. *window*, *is open*), and whether they connected them correctly (e.g. *window* to *is open*). Scores were calculated by subtracting incorrect from correct items, divided by total items necessary for the task. As in [4], participants could create longer or more elaborate programs without penalty, as long as the derived function was the same. In that case, the total item and connection count was adapted to their solution.

4 Results

4.1 Intelligibility

An ANOVA with mean intelligibility scores as dependent variable and the three interfaces as independent variables did not find significant differences in intelligibility ($F_{2,76}$=0.027, p=0.973). On average, participants selected two correct options (1.95, 2.10, 2.04 for form-filling, data-flow and grid-canvas) while selecting less than one incorrect one (0.40, 0.51, and 0.55 respectively). Intelligibility scores were distinctly different for the two different tasks (Table 4), implying that they indeed were different in their complexity and difficulty to comprehend as intended in the test design.

Correct answers were identified in similar ratios for the three prototypes (Fig. 2). Switching conjunctive triggers in the first intelligibility task (I1) was identified as correct slightly more often in the grid-canvas prototype, while the scenario combining the conjunctive and second disjunctive trigger (second intelligibility task, I2) was found correct less often in the form-filling prototype. Incorrect answers were chosen more diversely: That one trigger alone would suffice for intelligibility task 1 appeared correct to more participants in data-flow and grid canvas than in form-filling. That the action would lead to the trigger (reversal of rule) was, on the other hand, thought to be correct more often in form-filling and grid-canvas than in data-flow. For the second intelligibility task, participants chose the option that either of the disjunctive items would trigger the action without the conjunctive item more often in data-flow, while they accepted a change in trigger hierarchy more often in grid-canvas than in the other two prototypes.

4.2 Configuration & Programming

Configuration. As described in 3.3 Procedure and Measures**,** several measures were used to characterize the performance with and assessments of the different approaches. Results were analyzed using repeated-measures analysis of variance (ANOVA), with Bonferroni-corrected p-values to compensate for multiple application of the statistical

Table 4. Intelligibility scores as ratio %.

Task	Form-Filling	Data-Flow	Grid-Canvas	Total
I 1	70.09	69.23	67.52	68.95
I 2	38.46	41.88	42.74	41.03
Total	54.27	55.56	55.13	

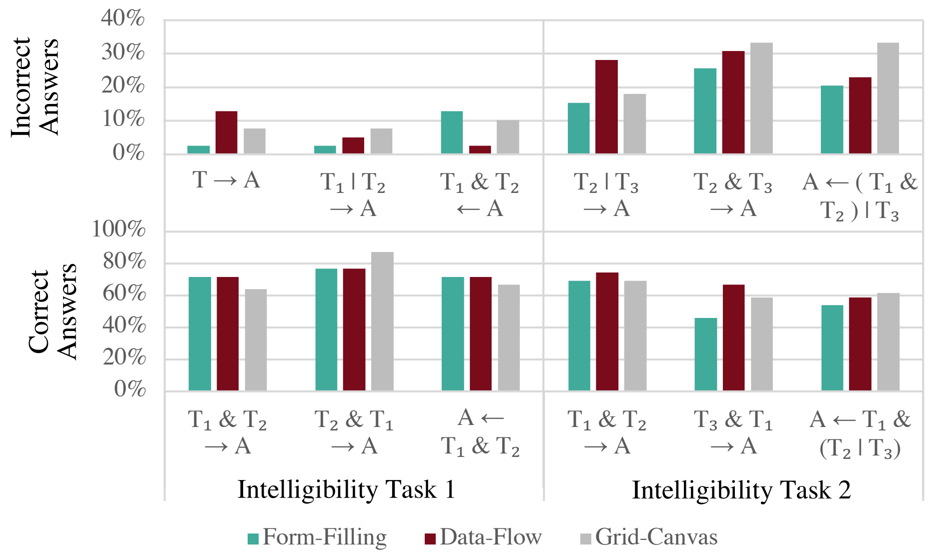

Fig. 2. Percentages of correct and incorrect answer choices in both intelligibility tasks.

testing. The assumption of sphericity was tested using Mauchly's test, and Greenhouse-Geisser correction of p-values was applied if required. The results of this analysis are summarized in Table 5. For the configuration task, we found significant differences in all measured dimensions except for *comprehension difficulty*. This is expected since the task descriptions were not prototype-specific and should, therefore, be comprehended equally. For the four other dimensions – *task completion time, task completion rate, perceived success and perceived programming* difficulty – the descriptive values show form-filling to perform best, closely followed by grid-canvas, and data-flow lagging clearly behind on each of the different measures.

Programming. Analyzing programming performance and experience employed the same procedure as the configuration task. A summary of the measurements and the statistical analysis is provided in Table 6. Overall a similar pattern to the configuration task can be observed, however, for programming only *task completion time* and *programming difficulty* show a significant difference between the three prototypes. Again, form-filling is fastest, while grid-canvas had the highest completion rate even though participants found it more difficult to program. Data-flow is remarkably slower and perceived as even more difficult.

Table 5. Configuration performance: descriptive measures for task completion time and rate, perceived success, perceived comprehension difficulty and perceived configuration difficulty.

	Mean (± Standard Deviation)			Repeated Measures - ANOVA
	Form-Filling	Data-Flow	Grid-Canvas	
Task Completion Time in seconds	75.66±49.47	130.0±104.4	88.32±84.11	$F_{2,66}$=26.45 [1] **p_{adj}<0.0001****
Task Completion Rate as ratio	0.922±0.157	0.874±0.143	0.915±0.160	$F_{2,76}$=5.801, **p_{adj}=0.0387***
Perceived Success 7-Point Likert Scale	6.542±0.881	6.054±1.507	6.304±1.250	$F_{2,76}$=6.032, **p_{adj}=0.0365***
Comprehension Difficulty 7-Point Likert Scale	1.603±0.933	1.756±1.051	1.673±0.999	$F_{2,76}$=1.120, p_{adj}=1
Configuration Difficulty 7-Point Likert Scale	2.016±1.444	2.974±1.851	2.093±1.385	$F_{2,76}$= 12.62, **p_{adj}=0.0001****

Table 6. Programming performance: descriptive measures for task completion time and rate, perceived success, perceived comprehension difficulty and perceived programming difficulty.

	Mean (± Standard Deviation)			Repeated Measures - ANOVA
	Form-Filling	Data-Flow	Grid-Canvas	
Task Completion Time in seconds	59.52±38.09	128.6±70.57	88.36±50.47	$F_{2,66}$=59.31 [1] **p_{adj}<0.0001***
Task Completion Rate as ratio	0.901±0.185	0.833±0.188	0.911±0.181	$F_{2,76}$=4.955, p_{adj}=0.0753
Perceived Success 7-Point Likert Scale	6.650±0.634	6.265±1.316	6.547±0.960	$F_{2,76}$=4.299, p_{adj}=1
Comprehension Difficulty 7-Point Likert Scale	1.385±0.717	1.479±0.714	1.530±0.761	$F_{2,76}$=1.186, p_{adj}=1
Programming Difficulty 7-Point Likert Scale	1.598±1.182	2.538±1.774	1.872±1.454	$F_{2,76}$=10.24, **p_{adj}=0.0012****

[1] Analysis for this factor is based only on data from 34 participants Due to a technical problem task completion times were not logged for the first 5 participants, therefore.

Performance & Complexity. To elucidate the impact of task complexity on performance with either of the prototypes, we analyzed *Total Completion Rate*, *Item Completion Rate*, and *Connection Completion Rate* per rule type (Table 7) [16]:

Mean *total completion scores* ranged from 80 to 99 percent in all three prototypes, with data-flow having a slightly lower range than the other two prototypes (81-92% as compared to 86-99% and 84-96% in form-filling and grid-canvas respectively). Form-filling had the highest *total completion rates* for substituting a trigger (C1), adding a disjunction and a trigger (C4), adding a disjunction, trigger and action (C5), and for adding an action with a time delay (C8). It also had the highest *completion rate* for programming the simplest task (P1). Data-flow meanwhile had the highest mean *total completion rate* for removing a trigger (C2) and for changing the trigger hierarchy (C7). Grid-canvas scored slightly higher than form-filling for adding a conjunctive trigger (C3), and noticeably higher than both other prototypes for changing a conjunction to a disjunction (C6) as well as for both remaining programming tasks (P2 and P3).

Mean *item completion scores* lay in similar ranges for all three prototypes (87-100%) and were nearly always higher than *connection scores* (70-99%), showing errors to be more frequent in the latter category. Differences between these means lay roughly between 0 and 20% for the form-filling and data-flow prototypes and between 0 and 10% for the grid-canvas prototype. Since grid-canvas did not perform better overall, the smaller range indicates that errors committed influences both item and connection completion jointly, while it influenced scores separately in the other prototypes.

Table 7. Total Completion Rates per Task.

Target rule structure		Total completion rate (item completion, connection completion)		
		Form-Filling	Data-Flow	Grid-Canvas
C 1	T → A	94.87% (97.44%, 92.31%)	89.57% (94.02%, 85.13%)	84.36% (89.23%, 79.49%)
C 2	T → A	86.41% (95.9%, 76.92%)	91.15% (91.28%, 91.03%)	90.73% (90%, 91.45%)
C 3	T & T → A	94.44% (96.58%, 92.31%)	83.24% (92.66%, 73.82%)	94.59% (95.33%, 93.85%)
C 4	T ǀ T → A	98.75% (98.78%, 98.72%)	83.41% (91%, 75.82%)	96.43% (96.63%, 96.24%)
C 5	T ǀ T → A & A	92.45% (92.59%, 92.31%)	80.67% (86.97%, 74.37%)	88.18% (88.95%, 87.41%)
C 6	T ǀ (T & T) → A	89.44% (97.86%, 81.03%)	88.7% (97.19%, 80.22%)	94.64% (98.11%, 91.17%)
C 7	T & (T ǀ T) → A	86.88% (98.89%, 74.87%)	91.58% (99.53%, 83.63%)	94.03% (96.69%, 91.38%)
C 8	T ǀ T → A & A → A	94.08% (95.33%, 92.82%)	90.62% (92.29%, 88.96%)	89.22% (90.82%, 87.61%)
P 1	T → A	96.79% (96.15%, 97.44%)	88.89% (95.73%, 82.05%)	92.76% (93.21%, 92.31%)
P 2	T & T → A	85.9% (91.89%, 79.91%)	80.68% (88.42%, 72.93%)	89.71% (90.18%, 89.23%)
P 3	T & (T ǀ T) → A	87.59% (93.64%, 81.54%)	80.36% (90.92%, 69.8%)	90.82% (94.32%, 87.32%)

Further investigation shows grid-canvas connection errors being based on incorrect connections between items (e.g. *window* and *living room*), while connection errors in the other prototypes were mostly made between rule structures (e.g. between two triggers). This is due to participants missing specifications more easily or placing them in a separate container in grid-canvas and frequently placing Boolean operators between triggers instead of at the end in data-flow. Meanwhile, trigger hierarchies were not understood well in either form-filling or grid-canvas. As anticipated, grid-canvas had better connection completion rates then data-flow except for configuration tasks 1 and 8. The former is due to the simplicity of the task, in which data-flow generally still performs well, the latter because participants frequently placed the time-delay action before a not-delayed action.

Comparison of configuration and programming. While the target configuration of tasks C1 and 2 corresponded to that of programming task 1, participants were more successful programming this simple rule from scratch than modifying it – programming had a 3% higher mean total completion score. Both other comparable tasks were configured more successfully than programmed, with an advantage of 5% and 4.5% (for tasks C3/P2 and C7/P3 respectively). Looking at the three prototypes (Fig. 3), we see that using data-flow, programming was less successful than configuring – slightly in the case of programming tasks 1 and 2, noticeably in programming task 3. In form-filling, programming P1 and P3 was more successful than modifying their equivalents C1/2 and C7, yet modification was more successful for C3, the equivalent of P2. In grid-canvas, participants had greater success programming P1 than configuring C1/C2, but higher scores for configuring C3 and C7 than for programming P2 and P3. The score differences for C1/P1 in grid-canvas and C2/P1 in form-filling are unexpected, indicating that substituting and removing a trigger might be more difficult in grid-canvas and form-filling respectively. That C7 was completed less successfully than the corresponding P3 in form-filling was likewise unexpected – participants might have missed trigger hierarchy settings more easily when checking a configuration than when programming. In grid-canvas, the difference between programming tasks 2/3 and their modification equivalents was similar, while in data-flow the difference was greater for C7 and P3 than for C3 and P2 – indicating that complexity affects programming more than modifying with data-flow.

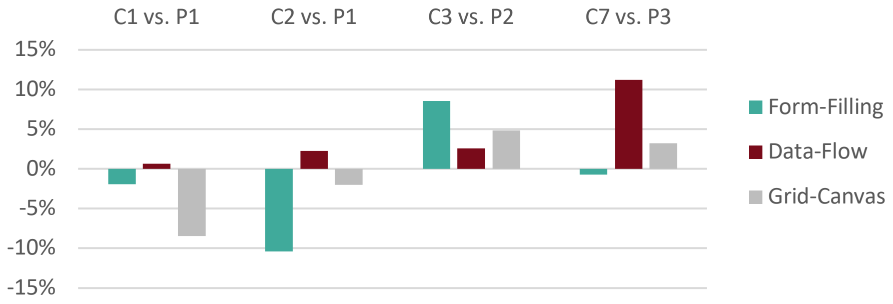

Fig. 3. Comparisons of total completion rates for configuration and programming tasks of identical target structures. Positive values indicate larger total completion scores in configuration than in corresponding programming tasks, negative values vice versa.

User Experience. Nearly half of the participants ranked form-filling as their favorite prototype, while only 21% and 31% did so for data-flow and grid-canvas. Grid-canvas was chosen for second place by 56%, while data-flow placed last for 69% of participants. Regarding scores from the User Experience Questionnaire (Fig. 4), form-filling and grid-canvas were experienced more similarly than data-flow. They both had higher *attractiveness*, *perspicuity*, *efficiency* and *dependability* scores than data-flow, which scored higher on *stimulation* and *novelty*. Participants highlighted the simplicity of form-filling and noted that it felt faster than other approaches, while also mentioning that it was less easy to comprehend at a glance and in need of visual structuring. They indicated data-flow to give a good overview, especially regarding its a central repository and trigger hierarchies, as well as a greater sense of freedom, creativity, and invitation. As such, they also noted that is too little structured to be functional, and too time-intensive. They appreciated the split of the canvas in grid-canvas prototype with its dedicated *if/then* layers, which combined a good overview with greater clarity, but also noted the individual drag-and-drop as cumbersome, suggesting an automated population of specification options once a main element was moved into a container. Participants who had noted the existence of trigger hierarchy markers in form-filling and grid-canvas also mentioned that they did not fully understand their impact.

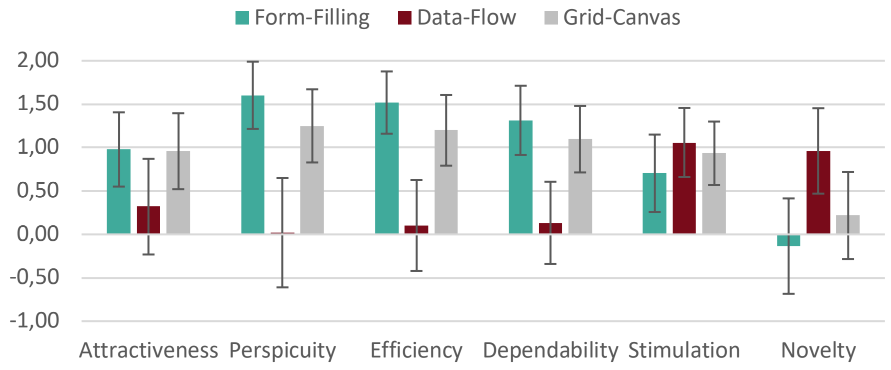

Fig. 4. Mean scores of the User Experience Questionnaire with 95% Confidence Interval. Scores ranged from -3 to 3 on each dimension.

5 Discussion

Within this study, we compared three different approaches to end-user programming in a home context with regard to their usefulness and understandability. While the use of prototypes poses its own limitations, especially regarding their maturity, it is currently the only option to compare approaches that do not differ in complexity, cut-off for expressiveness, vocabulary, or visual polish.

Our study shows **significant differences between prototypes,** specifically in *task completion time*, *rate*, *perceived success*, and *perceived configuration difficulty* in the configuration task, and in *task completion time* and *perceived programming difficulty* in the programming task. Form-filling is shown to be faster for both modification and programming, even though programming was slightly more successful using the grid-canvas prototype. Participants had higher confidence in their solution in form-filling and rated its use as slightly easier than grid-canvas. This indicates that grid-canvas could benefit from guiding trigger selection (e.g. by automatically adding necessary specification fields once the main trigger was chosen). The biggest difference is however between these two prototypes and data-flow which was considerably slower, had lower *task completion scores* was perceived as more difficult and less successful. However, it had advantages when removing a trigger and changing trigger hierarchies. Grid-canvas out-performed the other prototypes in changing Boolean operator types and programming all but the simplest rule from scratch. It is notable that the difference between item and connection scores is lower in the grid-canvas prototype, showing that errors committed influence both item and connection completion jointly. This indicates that while the form-filling and data-flow prototypes need to improve their support of forming connections (including choice of trigger hierarchies), grid-canvas would benefit from supporting item choice. Specifically, this includes indicating items or specifications missing from a container, enabling users to move entire containers and visualizing that time delay impacts all following actions. Other prototype improvements include greater support for trigger hierarchies in form-filling and grid-canvas approaches and structural adjustment of data-flow regarding Boolean operator placement (between triggers instead after them).

Regarding scores from the User Experience Questionnaire, differences between form-filling and data-flow were in general much similar to the results in [4]: Form-filling was rated as more attractive, perspicuous, efficient, and dependable but less stimulating and less novel. Form-filling was even rated as less attractive and less novel in this study then it did in [4]. There was a minor improvement of the data-flow prototype in perspicuity, but slightly lower values for efficiency and dependability. The new grid-canvas prototype was rated as attractive as form-filling and received slightly lower scores in perspicuity, efficiency, dependability while scoring higher on stimulation and novelty. Qualitative comments, for the most part, echoed these User Experience Questionnaire scores. It is especially interesting that participants mentioned feelings of freedom and restriction respectively regarding the data-flow and form-filling prototypes, which is much in line with results in [3] and suggests an investigation of the role of creativity and freedom in practical smart home appliances.

Apart from elucidating differences between end-user programming approaches, our study also proposes three additions to Caivano et al.'s [17] ten **design implications** for the creation of event-condition-action rules:

1. **Facilitate rule intelligibility**
   Employing two different intelligibility tasks, we found no significant difference between the prototypes, yet their generally low understandability shows that there is a need to investigate how novices understand end-user approaches and to

develop methods with which to impart and measure understanding. Correct and incorrect answer choices indicate strengths and weaknesses for each prototype regarding how well users understand which triggers are necessary, their order and hierarchy as well as general rule direction, demonstrating the need to individually consider these aspects in supporting systems.

2. **Facilitate complex rules: trigger connections and hierarchies**
As in a previous study [4, 16], the total completion rate did not simply decrease with rule complexity, showing that is it is not the number of items but rather the number of triggers and the complexity of their connections (conjunctions, disjunctions as well as hierarchies) that impact configuration and programming performance. Approaches that offer a guided configuration experience [e.g. 11] or visualize differences between program options [e.g. 18] show potential to significantly ease this, but have yet to be tested with programs that use disjunctions and trigger hierarchies. Visualizing differences between program options would additionally need to be embedded within the programming process to be used in this manner.

3. **Support modification additionally to programming from scratch**
Our study shows that complexity influences programming and modifying with a prototype differently. Comparable target rules show a slightly better total completion score for modifying instead of programming more complex rules from scratch. This illustrates the importance of predefined patterns to modify by end-users e.g. by supporting re-use [17] as well as of creating an interface that does not merely lend itself to programming from scratch but also to modification.

Since individual preferences for analytical thinking or experiential engagement as well as for visual and verbal processing could impact interface preferences [30, 31], future work will also include such personal characteristics.

Acknowledgments. This research was performed within the SIM4BLOCKS project (funded from the European Union's Horizon 2020 research innovation program under grant agreement No. 695965).

References

1. Reisinger M.R., Prost S., Schrammel J., Fröhlich P.: User Requirements for the Design of Smart Homes: Dimensions and Goals in Chatzigiannakis, I., De Ruyter, B., and Mavrommati, I. (eds.) Ambient Intelligence. Lecture Notes in Computer Science, vol 11912. pp. 41–57. Springer, Cham (2019)
2. Davidoff S., Lee M.K., Yiu C., Zimmerman J., Dey A.K.: Principles of smart home control in Dourish, P. and Friday, A. (eds.) UbiComp 2006. vol. 4206. pp. 19–34. Springer-Verlag Berlin Heidelberg (2006)
3. Brich J., Walch M., Rietzler M., Weber M., Schaub F.: Exploring End User Programming Needs in Home Automation ACM Trans. Comput. Interact., 24, pp. 1–35 (2017)
4. Reisinger M.R., Schrammel J., Fröhlich P.: Visual languages for smart spaces: End-user

programming between data-flow and form-filling 2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC). pp. 165–169. IEEE (2017)
5. Rode J.A., Toye E., Blackwell A.: The fuzzy felt ethnography - understanding the programming patterns of domestic appliances Pers. Ubiquitous Comput., 8, pp. 161–176 (2004)
6. Lucci G., Paternò F.: Understanding End-User Development of Context-Dependent Applications in Smartphones 5th IFIP WG 13.2 International Conference of Human-Centered Software Engineering - HCSE 2014. vol. 8742. pp. 182–198 (2014)
7. Lucci G., Paternò F.: Analysing How Users Prefer to Model Contextual Event-Action Behaviours in Their Smartphones 5th International Symposium on End-User Development, IS-EUD 2015. vol. 9083. pp. 186–191 (2015)
8. Ur B., McManus E., Pak Yong Ho M., Littman M.L.: Practical trigger-action programming in the smart home Proc. 32nd Annu. ACM Conf. Hum. factors Comput. Syst. - CHI '14, pp. 803–812 (2014)
9. Paternò F., Santoro C.: New Perspectives in End-User Development pp. 43–59 (2017)
10. Dey A.K., Sohn T., Streng S., Kodama J.: iCAP: Interactive Prototyping of Context-Aware Applications 4th International Conference, PERVASIVE 2006. vol. 3968 LNCS. pp. 254–271 (2006)
11. Fogli D., Peroni M., Stefini C.: ImAtHome: Making trigger-action programming easy and fun J. Vis. Lang. Comput., 42, pp. 60–75 (2017)
12. Ur B., Pak M., Ho Y., Brawner S., Lee J., Mennicken S., Picard N., Schulze D., Littman M.L.: Trigger-Action Programming in the Wild : An Analysis of 200 , 000 IFTTT Recipes Proc. SIGCHI Conf. Hum. Factors Comput. Syst., pp. 3227–3231 (2016)
13. De Russis L., Corno F.: HomeRules CHI EA '15 - Proceedings of the 33rd Annual ACM Conference Extended Abstracts on Human Factors in Computing Systems. pp. 2109–2114. ACM Press, New York, New York, USA (2015)
14. Cabitza F., Fogli D., Lanzilotti R., Piccinno A.: Rule-based tools for the configuration of ambient intelligence systems: a comparative user study Multimed. Tools Appl., 76, pp. 5221–5241 (2017)
15. Cabitza F., Fogli D., Lanzilotti R., Piccinno A.: End-User Development in Ambient Intelligence : a User Study CHItaly 2015 Proceedings of the 11th Biannual Conference on Italian SIGCHI. pp. 146–153 (2015)
16. Reisinger M.R., Schrammel J., Fröhlich P.: Visual end-user programming in smart homes: Complexity and performance 2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC). pp. 331–332. IEEE (2017)
17. Caivano D., Fogli D., Lanzilotti R., Piccinno A., Cassano F.: Supporting end users to control their smart home: design implications from a literature review and an empirical investigation J. Syst. Softw., 144, pp. 295–313 (2018)
18. Zhao V., Zhang L., Wang B., Lu S., Ur B.: Visualizing Differences to Improve End-User Understanding of Trigger-Action Programs pp. 1–10 (2020)
19. Corno F., Russis L.D.E., Roffarello A.M.: RecRules : Recommending IF-THEN Rules for End-User Development 10, (2019)
20. Corno F., De Russis L., Monge Roffarello A.: My IoT Puzzle: Debugging IF-THEN Rules Through the Jigsaw Metaphor Presented at the (2019)
21. Dahl Y., Svendsen R.-M.: End-User Composition Interfaces for Smart Environments: A Preliminary Study of Usability Factors 1st International Conference on Design, User Experience and Usability, DUXU 2011, Held as Part of HCI International 2011. vol. 6770. pp. 118–127 (2011)
22. Wiedenbeck S., Engebretson A.: Comprehension strategies of end-user programmers in an event-driven application Proc. - 2004 IEEE Symp. Vis. Lang. Hum. Centric Comput., pp. 207–214 (2004)
23. Collar E., Valerdi R.: Role of Software Readability on Software Development Cost Proc.

21st Forum COCOMO Softw. Cost Model., (2006)
24. Lee T., Lee J.B., In H.P.: A study of different coding styles affecting code readability Int. J. Softw. Eng. its Appl., 7, pp. 413–422 (2013)
25. Relf P.A.: Achieving Software Quality through Source Code Readability Qualcon 2004 (2004)
26. Jahnke J.H., D'Entremont M., Stier J.: Facilitating the programming of the smart home IEEE Wirel. Commun., (2002)
27. Abdallah R., Xu L., Shi W.: Lessons and experiences of a DIY smart home Proceedings of the Workshop on Smart Internet of Things - SmartIoT '17. pp. 1–6. ACM Press, New York, New York, USA (2017)
28. Davidoff S., Lee M., Zimmerman J., Dey A.: Socially-aware requirements for a smart home IE '06 - Proceedings of the international symposium on intelligent environments. pp. 41–44. Citeseer (2006)
29. Laugwitz B., Held T., Schrepp M.: Construction and Evaluation of a User Experience Questionnaire HCI Usability Educ. Work, pp. 63–76 (2008)
30. Mancy R., Reid N.: Aspects of cognitive style and programming 16th Work. Psychol. Program. Interes. Gr., pp. 1–9 (2004)
31. Lee J.: The effects of visual metaphor and cognitive style for mental modeling in a hypermedia-based environment Interact. Comput., 19, pp. 614–629 (2007)

Appendix A: Task Sets

Translated by the author. Please note that translation can introduce discrepancies not present in the original text. For example, *if* and *when* are not differentiated in German. Events and conditions are linguistically more easily distinguishable – this is mimicked in the translation by using *is on/off* for the state or condition, while using *is turned on/off* for the event. Likewise, some features might be expressed differently in each language.

Table A.1. Intelligibility tasks: Configurations per task set.

Task	Rule Configuration Taskset 1	Rule Configuration Taskset 2	Rule Configuration Taskset 3
I 1	W-Lan (off) & alarm system (on) → TV (living room, off)	Door (kitchen, closed) & light (kitchen, off) → blinds (kitchen, close)	Doorbell (rings) & coffee machine (on) → light (kitchen, dim)
I 2	Ventilation (on) & (humidifier (on) | window (home, open) → message (send)	Wind detector (active) & (window (home, open) | blinds (home, closed) → message (send)	Light (living room, on) & (sun setting | rain) → light (living room, warm)

Table A.2. Intelligibility tasks: Answer structure and options per task sets.

Task	Answer Structure[a]	Correct	Taskset 1	Taskset 2	Taskset 3
I 1	$T_1 \rightarrow A$		If the W-Lan is off, the TV in the living room is switched off.	If the kitchen door is closed, the blinds in the kitchen close.	If the doorbell rings, the light in the kitchen dims.
	$T_2 \ \& \ T_1 \rightarrow A$	x	If the alarm system is on, the TV in the living room is turned off, provided that the W-Lan is off.	If the light in the kitchen is off, the blinds in the kitchen will close, provided the kitchen door is closed.	If the coffee machine is on, the light in the kitchen dims, provided the doorbell rings.
	$T_1 \ \& \ T_2 \rightarrow A$	x	If both the W-Lan is off and the alarm system is on, the TV in the living room is turned off	If both the kitchen door is closed and the kitchen light is off, the blinds in the kitchen close	If both the doorbell rings and the coffee machine is on, the light in the kitchen dims.

Task	Answer Structure[a]	Correct	Taskset 1	Taskset 2	Taskset 3
	$T_1 \mid T_2 \rightarrow A$		If either the W-Lan is off or the alarm system is on, the TV in the living room is turned off.	If either the kitchen door is closed or the kitchen light is switched off, the blinds in the kitchen close.	If the doorbell rings or the coffee machine is on, the light in the kitchen dims.
	$A \leftarrow T_1 \& T_2$	x	The TV in the living room is turned off because the alarm system is on and the W-Lan is off.	The blinds in the kitchen close because the kitchen light is off, and the kitchen door is closed.	The light in the kitchen dims because the coffee machine is on and the doorbell rings.
	$T_1 \& T_2 \leftarrow A$		The W-Lan is off, and the alarm system is on because the TV in the living room is switched off.	The kitchen light is off, and the kitchen door closed because the blinds in the kitchen are closed	The doorbell rings and the coffee machine are on because the kitchen light is dimmed.
	$T_2 \mid T_3 \rightarrow A$		If the ventilation or humidifier is on, a message is sent.	If a window in the apartment is open or a blind in the apartment is closed, a message is sent.	If the sun is setting or it is raining, the living room light changes its color temperature to warm.
	$T_1 \& T_2 \rightarrow A$	x	If a window in the apartment is open and the ventilation is on, a message is sent.	If the wind detector is active and a window in the apartment is open, a message is sent.	If the living room light is on and the sun is setting, the living room light changes its color temperature to warm.
	$T_3 \& T_1 \rightarrow A$	x	If the humidifier is on and a window in the apartment is open, a message is sent.	If a blind in the apartment is closed and the wind detector is active, a message is sent.	If it is raining and the living room light is switched on, the living room light changes its color temperature to warm.
I 2	$T_2 \& T_3 \rightarrow A$		If the ventilation and humidifier are on, a message is sent.	If a window in the apartment is open and a blind in the apartment is closed, a message is sent.	If the sun is setting and it is raining, the living room light changes its color temperature to warm.
	$A \leftarrow T_1 \& (T_2 \mid T_3)$	x	A message is sent because a window in the apartment is open and either the ventilation or the humidifier was on.	A message is sent because the wind detector is active and either a window in the apartment is open or a blind in the apartment is closed.	The light in the living room has changed its color temperature to warm because the living room light is on and either the sun is setting, or it is raining.
	$A \leftarrow (T_1 \& T_2) \mid T_3$		A message is sent because either a window in the apartment is open and the ventilation is on, or because the humidifier is on.	A message is sent because either the wind detector is active and a window in the apartment is open, or because a blind in the apartment is closed.	The living room light changes its color temperature to warm, because either the living room light is on and the sun is setting, or because it is raining.

[a]. T indicates a trigger, A an action, & a conjunction, | a disjunction

Table A.3. Configuration Tasks: Initial and target configuration descriptions of all three task sets.

Task	Initial Configuration [a]	Target Configuration Instruction
	Time (18:30) → W-Lan (turn on)	If someone is in the living room, turn on the W-Lan.
C 1	Time (18:00-23:00) → humidifier (turn on)	If I am at home, turn on the humidifier.
	Time (7:30-18:00) → alarm system (turn on)	If nobody is at home, turn on the alarm system.
	Someone (kitchen) & blinds (kitchen, closed) → stereo (kitchen, turn on)	If someone is in the kitchen, the stereo in the kitchen should be turned on.
C 2	I (bathroom) & light (bathroom, is turned on) → stereo (bathroom, turn on)	If I am in the bathroom, the stereo in the bathroom should be turned on.
	Blinds (living room, closed) & nobody (living room) → TV (living room, turn off)	If nobody is in the living room, the TV in the living room should be turned off.
	Doorbell → lights (anteroom, turn on)	If the doorbell rings and someone is at home, the light in the anteroom should turn on.
C 3	Sun (setting) → light (living room, dim)	If the sun is setting and nobody is in the apartment, the light in the living room should be dimmed.
	Alarm clock → light (bedroom, turn on)	If the alarm clock rings and I am at home, the light in the bedroom should turn on.
	Smoke detector (active) → message (send)	If the smoke detector or the alarm system is active, a message should be sent.
C 4	CO_2 (low) → ventilation (turn off)	If the CO_2 level or humidity is low, the ventilation should switch off.
	Wind detector (active) → blinds (home, open)	If the wind detector or smoke detector is active, the blinds should open in the entire apartment.
	Door (living room, open) → humidifier (turn off)	If the door or window in the living room is open, the humidifier should turn off and the blinds in the living room should be opened.
C 5	Light (living room, is turned on) → W-Lan (turn on)	If the light in the living room is turned on or the blinds are opened in the living room, the W-Lan and TV in the living room should turn on.
	Humidifier (on) → ventilation (turn off)	If the humidifier is on or the blinds are opened in the kitchen, the ventilation should turn off and the light color in the kitchen should change to cool.
C 6	Humidity (high) & CO_2 (high) & ventilation (on) → humidifier (turn off)	If the humidity is high or If both the CO_2 level is high and the ventilation is on, then the humidifier should be turned off.
	Smoke detector (off) & alarm system (off) & W-Lan (on) → message (send)	If the smoke detector is off or both the alarm system is off and the W-Lan is on, a message should be sent.

Task	Initial Configuration [a]	Target Configuration Instruction
C 7	Coffee machine (off) & humidity (low) & ventilation (off) → humidifier (turn on)	If the coffee machine is off or both the humidity is low and the ventilation is off, the humidifier should switch on.
	(Nobody (home) & window (home, is opened)) ∣ door (home, is opened) → alarm system (alarm)	If nobody is at home and either windows or doors are opened in the apartment, the alarm system should sound the alarm.
	(I (home) & stereo (bedroom, on)) ∣ door (bedroom, is opened) → coffee machine (turn on)	If I am at home and either the stereo in the bedroom is on or the door to the bedroom is opened, the coffee machine should turn on.
	(Someone (living room) & TV (living room, on)) ∣ light (living room, on) → W-Lan (turn on)	If someone is in the living room and either the TV or the light in the living room is on, the W-Lan should be switched on.
C 8	Switch (bedroom) ∣ Time (7:00) → Heating (set to 22°C) & coffee machine (turn on)	If the switch in the bedroom is pressed or it is 7:00, the heating should be set to 22°C and the light color in the bedroom should change to warm. After 5 minutes the coffee machine should switch on.
	Switch (anteroom) ∣ distance from home (<400 m) → Heating (set to 22°C) & W-Lan (turn on)	If the switch at the entrance is pressed or I am less than 400 meters away, the heating should be set to 22°C and the light in the anteroom should be turned on. After 5 minutes the W-Lan should switch on.
	Switch (anteroom) ∣ distance from home (>500 m) → Heating (set to 18°C) & alarm system (turn on)	If the switch at the entrance is pressed or I am more than 500 meters away, the heating should be set to 18°C and all lights should be turned off. After 5 minutes the alarm system should turn on.

Table A.4. Programming Tasks: Target rule structures and configuration instruction of all three task sets.

Task	Target Rule [a]	Target Configuration Instruction
P 1	T → A	If someone is at home, turn off the alarm system.
		If nobody is in the kitchen, the coffee machine should turn off.
		If I'm in the living room, turn on the air conditioning.
P 2	T & T → A	If the sun is setting and nobody is in the living room, the blinds in the living room should close.
		If the alarm clock rings and I am in the bedroom, the stereo in the bedroom should turn on.
		If it rains and someone is in the kitchen, the light color in the kitchen should change to cool.
P 3	T & (T ∣ T) → A	If nobody is in the bathroom and either the door or window in the bathroom is closed, the ventilation should turn on.
		If I am at home and either the TV or stereo system in the living room is turned on, the W-Lan should turn on.
		If someone is in the kitchen and either the stereo or TV in the kitchen is on, the coffee maker should turn on.

[a]. T indicates a trigger, A an action, & a conjunction, ∣ a disjunction