Towards Explainable Automations in Smart Homes Using Mobile Augmented Reality

Andrea Mattioli [1,2] and Fabio Paternò [1]

[1] CNR-ISTI, HIIS Laboratory, Pisa, Italy
[2] Department of Information Engineering, University of Pisa, Pisa, Italy

Abstract

The pervasiveness in daily environments of objects equipped with sensors and actuators and characterized by the possibility of communicating over the Internet has steadily increased in recent years. In this scenario, smart home automations are becoming increasingly adopted. It is hence important to provide users with explainable tools to better control these automations and make them more useful for their needs. We present a novel mobile augmented reality solution to support users in creating and controlling automations through recommendations and a simulation tool. We also discuss the application of an augmented reality XAI framework to the presented solution in order to improve its transparency.

Keywords

Augmented Reality, Explanation, Personalization

1. Introduction

In automation-based environments it is possible to have several automations active at the same time with a resulting behaviour different from the expected one. In this case, explainability strategies should be put in place to identify potential problems, and help in finding possible solutions, involving users in this process. One possible strategy is to provide explanations for contributing to reaching some overall goal for the targeted context of use. Examples of possible overall goals are improving security or health or energy saving or entertainment. A key point is how people can receive useful information and help in identifying possible mismatches between desired and actual behaviour. A way to reduce the likelihood of errors in the specification of automations is to allow users to simulate the conditions and events that can trigger an automation and the effects that they will bring about. Alternatively, automations could be actually applied and executed in the current context of use and explain how they work. While testing is aimed at detecting the presence of errors in the programs built by users, debugging is the process of finding the cause of the identified errors in the behaviour and fixing/removing them. We intend to support both. This can be quite problematic for end users especially because, as noted by [6], most EUD environments do not include debugging aids for unprofessional end users. A general approach for the debugging part is represented by the Interrogative Debugging paradigm [10], in which the system directly answers "why" and "why not" questions. In this perspective we have considered previous studies on the use of why and why not explanations to improve the intelligibility of context-aware intelligent systems [12]. More recently, general approaches for explainable artificial intelligence have been put forward with the goal to identify the key questions to address for this purpose [11]. Such questions have been refined for the augmented reality context [14]. Even such recent contributions have not addressed the issue of how to provide useful explanations about the available automations through a mobile augmented reality solution. In this paper we introduce a novel solution to effectively localize issues in the currently specified behaviours, and provide users with support through natural-language explanations of why or why not the automations can/cannot be

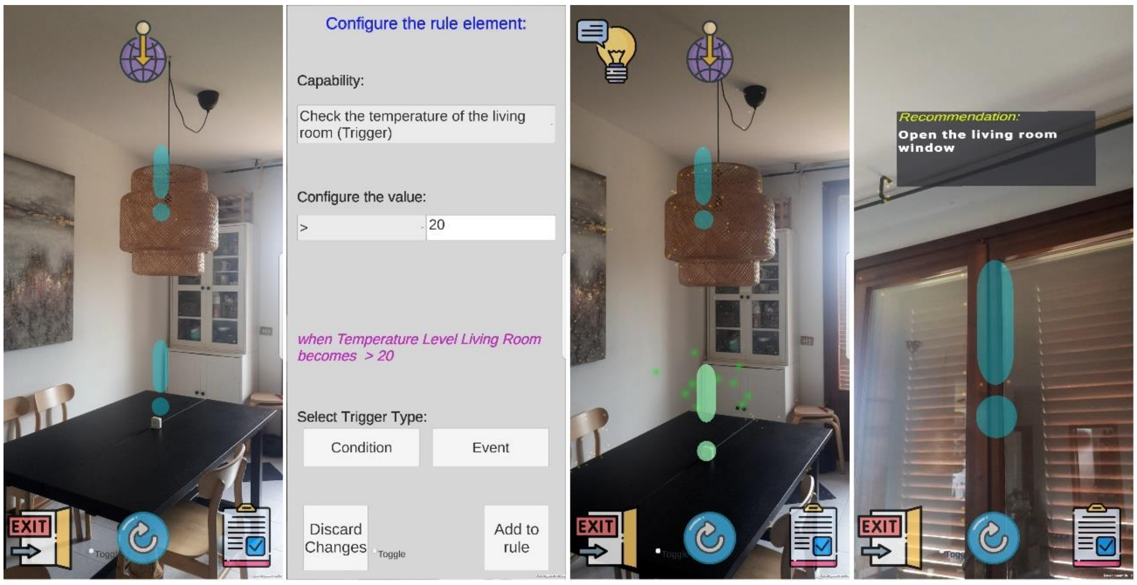
Proceedings Name, Month XX–XX, YYYY, City, Country
EMAIL: andrea.mattioli@isti.cnr.it (A. Mattioli); fabio.paterno@isti.cnr.it (F. Paternò)
ORCID: 0000-0001-6766-7916 (A. Mattioli); 0000-0001-8355-6909 (F. Paternò)

correctly executed, moreover accompanied by concrete examples (or counterexamples) highlighting the situations in which a specific automation is verified or not. Another way to provide support to users is to show recommendations that match the intended behaviour they are currently defining. Previous work [3, 5, 13, 15] analysed how to generate and present automation recommendations in IoT scenarios. However, how to generate, present, and explain recommendations using a mobile augmented reality approach is an untapped and potentially fruitful new direction.

In particular, this paper presents two main contributions. The first is the introduction of the ARACS (Augmented Reality Automation Creation and Simulation) platform (Andrea Mattioli and Fabio Paternò, A Mobile Augmented Reality App for Creating, Controlling, Recommending Automations in Smart Homes, submitted paper), an effort to empower users to better understand, configure, and modify automations in their everyday environments with the support of recommendations and a context simulator/debugger. The second is a discussion about how to improve the transparency of the ARACS system through the application of an augmented reality explainable AI framework.

2. Augmented Reality Automation Creation and Simulation

The proposed platform is an Android application developed in Unity using the ARFoundation library. The application exploits the camera of the mobile phone to capture the user's environment, and place visualization over the physical objects available in the current context which functionalities can be used to define automations. The main functionalities of the application are "Create automation" and "Explore environment". The first allows users to personalise the environment with automation rules involving objects and services while moving about in their spaces. The configuration of the automation (see Figure 1) is performed in a situated and dynamic way: at the start of the application, a visualisation is placed over the objects that can be used in the automations. Selecting one of these visualisations, a panel describing the functionality that can be used as a trigger or action is shown. After the configuration, the visualisation over the object changes, indicating its use in the current automation. Also, panels with recommendations related to the inserted configuration are placed over the other objects. The user can then continue the visit, moving to the next object she wants to use in the automation and configuring it, repeating this process until the automation is completed. Supporting information can be presented on request, for instance showing the partial automation configuration done at the current moment. The "Explore environment" functionality is used to make the relation between the automations and the real objects more transparent and perceivable. When using this functionality, natural language descriptions of the rules defined by the user are displayed over the corresponding objects.


Configure the rule element:

Capability:

Check the temperature of the living room (Trigger)

Configure the value:

> 20

when Temperature Level Living Room becomes > 20

Select Trigger Type:

Recommendation:

Open the living room window

Figure 1: Example of the "Create automation" functionality. (Left): visualizations are placed over the objects that can be used in automations. (Left-centre): after selecting the visualization over the multipurpose sensor, a panel to configure it is shown. (Right-centre): after pressing "add to rule", the

configuration is added to the automation the user is currently defining, and the visualization changes to indicate it. (Right): recommendations related to the inserted configuration are placed over the other objects, in the example, the action to automatically open the window.

The platform provides two types of intelligent support. The first is rule elements recommendations. In the context of EUD for smart home configuration, recommendations should help users to define the joint behaviours of the IoT devices and services. Hence, the system should present users with recommendations during the configuration of these behaviours, providing diverse options to complete them. The adopted approach is to generate personalised recommendations starting from a dataset of automation rules[2]. The recommender system matches the user representation and what she is currently defining (the recommendation context, represented using one-hot encoding) with the automations in the dataset, also leveraging the similarity between the natural language transcription of the context and the textual parts of automations defined by other users. The textual similarity match is performed using a large language model, BERT [7]. This recommendations approach has been implemented using neural collaborative filtering [8]. This deep learning architecture leverages neural networks to model the interaction between the users and the items to suggest, and it is hence capable of learning complex non-linear relationships between them. Since the various inputs are projected in a vector space, the embeddings generated by a large langue model can also be used in the process. This approach is hence a generalization of the matrix factorization-based collaborative filtering and is capable of learning from different inputs. The output of the model is a score indicating how well a rule element to recommend fits with the various inputs provided (in this case, the user representation, the one-hot context representations, and its natural language description).

The second type, which is currently under development, is a rule simulation and debugger. The rule simulation is an extension of the "Explore environment" functionality, to consider and visualise also the state of the environment (the context). It will allow loading the current context or some predefined context snapshots (for instance, summer morning, weekday night, dinner with friends), and visualising the associated values over the various object in the environment. The users can modify these contextual values (for instance, changing the time, or acting on the temperature) to check whether some automation will or will not activate in the environment, providing them a "Why/Why not" [10, 12] supporting tool. Furthermore, the possibly problematic relations between automation rules will be made perceivable. As reported in previous work [4, 9, 16] and summarised in [1], three main unexpected relations between automations can lead to logical errors, namely, rule prevention (when the execution of a rule prevents the triggering of another one), rule collisions (when the outcome of two rules are in contrast), and unexpected rule chains (when the activation of a rule cause another one to trigger). In general, to detect these problems an intermediate graph representation of the automations active in the system (hence that will be executed if the triggering conditions are met) is needed. An approach to detect these errors and make them perceivable in the environment will be developed together with the rule simulation.

3. Application of the XAIR Framework to the ARACS Platform

In the following, the XAIR framework, its key factors, and the steps to assess them will be introduced. Then, it will be analysed an application of the framework to the ARACS platform to determine how to present AR explanations in two use-case scenarios.

3.1. XAIR Framework

We based the design of the augmented reality explanations on the XAIR design framework [14]. The goal of XAIR is to support the design of effective XAI experiences for AR, addressing the key concepts needed to provide explanations of AI output. The framework is the result of an analysis of the literature, a large-scale end-user survey, and workshop iterations with designers and experts in relevant fields. It is based on a definition of the problem space (when, what, and how to explain) and the assessment of the key factors needed to determine the answers to these questions. The design space is structured as

[1] https://github.com/andrematt/trigger_action_rules, last accessed 2023/05/10.

follows. On the **“When”** dimension, the two main aspects are the availability of the explanation (whether the explanation should be always ready or not) and the delivery (when to show it). Since the availability should always be ready to improve user experience, the decisions are about the delivery timing. An explanation can be automatically presented or shown when requested by the user, and both approaches can be valid depending on the use case. The second dimension is the **“What”**. It articulates in content and detail level. About the content, seven types of explainable content were identified: input/output, why/why not, how, certainly, example, what if, and how to, delivered possibly via local explanation. Concerning the detail, the priority is to provide explanations that expand users’ prior knowledge or fulfil their immediate needs, also considering the cognitive capability available and the possibility of presenting personalized and detailed information. About the **“How”** dimension, the identified levels are the modality (visual and audio emerge as the most suited) and the paradigm, where the larger design space concerns the visual modality. The design aspects to consider in the visual modality are the format (textual and graphical, eventually combined) and the pattern type (implicit, explicit). Implicit pattern refers to naturally blending the additional information with the referring object, for instance directly highlighting an anomaly with a circle or an arrow over it, while explicit refers for instance to an extended dialogue window.

The second part of the framework consists of the steps to assess the key factors in relation to the defined dimensions. For the **“When”** dimension, the decision is about when to present the auto-triggered explanation (by default, the explanation should be on-demand). The first necessary condition is that the user has enough cognitive capability and time to engage with the explanation. The second is that at least one of the conditions is true among 1) there is a mismatch between the user expectation and the received recommendation, hence, the user is surprised or confused; 2) the user is not familiar with the output of the system; and 3) the system is uncertain of the output. Concerning the **“What”**, the explanations should be contextualised considering the user goal (depending on what the user is doing, a type of explanation can be more useful than another), the system goal (for instance, a recommendation can be shown to calibrate the system to the user taste, to propose alternatives, to help manage errors, or to improve trust in the system), and the user profile (in particular the AI literacy). Regarding the detail level, by default explanations should be short and focused on the why aspect, with the possibility of providing more details upon user request. For the **“How”** part, the modality should normally be the same as the outcome of the AI process, but for specific situations (e.g., when the used channel is overloaded) an alternative one should be used. Concerning the paradigm of visual explanations, textual should be the main explanation format, and a simple graphic such as icons could provide additional information. More advanced graphics (such as heatmaps) could be used in detailed explanations. About the pattern, when possible the explanation should be implicit (blended with the real environment), while the explicit possibility represents the backup solution.

3.2. Recommendations during rule creation

The first scenario involves the rule recommendation functionality of the ARACS platform. The user wants to automate the morning air circulation in the living room. She configures the time the automation should activate, and then the living room window opening action. After these operations, the recommendation “if (condition) the humidity level of the room is more than 70%” is placed in a panel over the multipurpose sensor in the living room, indicating that the humidity check functionality can be used together with widows automatic opening and time checks. The user is surprised since she was thinking about temperature or weather-related recommendations, but not humidity.

Following the XAIR framework, the first dimension to assess is the **“When”**. Since the user is currently defining the automation, we can assume that she has enough cognitive capability and time to eventually engage with explanations. Then, an assessment is performed to establish whether a condition is true among the three necessary for targeting the explanation. In this scenario, possible cases for each condition can verify. For instance, the user can receive a suggestion of a rule whose main goal is different from her intended goal for that automation, e.g., security instead of energy saving, or it concerns a service she had never used before. In this case, the condition for automatically presenting an explanation is true because she has never used the “humidity level” trigger. Indeed she's surprised because she wasn't even aware that the sensor had that feature. Concerning the **“What”** part, the first assessment is about the System goal, which is user intent discovery (suggesting a user with new automation possibilities that she may find useful and be unaware of). The main user goal is to resolve the surprise since she received an unexpected recommendation. The last assessment is about the user AI literacy, which we know from the user profile to be low. Hence, the framework's suggested explanation types for the intersection of the various parameters are Input/Output and Why/Why-not explanations. Concerning the detail level, the interface can show the why as default (for instance, with a text explaining that humidity checks are often used to automate the air circulation), and the Input/Output can be shown as additional information. Concerning the **"How"**, the modality of the explanation is visual, using text as a default. The detailed explanation can show the information that impacted the decision to recommend this automation part (e.g., which part of the incomplete automation inserted by the user had more weight for proposing the recommendation, or which aspect of the user profile), or how the recommendation changes when some parameter is modified (for instance, the automation goal, or using a default user instead of the current one). The pattern can be explicit (text panels), with some implicit components, for instance highlighting with a bright colour the objects with positive impact for a specific recommendation.

3.3. Situated automations simulation and debugging

The second scenario concern using the environment simulator/rule debugger to better understand the smart environment behaviour and eventually help the user to solve logical errors in the configured automations. In this scenario, the user has configured the smart coffee maker to brew a coffee at the time she wakes up, at 6:30 AM. In the morning, she finds out that the machine is on, but no coffee has been brewed. To debug the situation, she loads the most similar environment predefined conditions, namely the "night weekday" preset. Once loaded, the augmented representations over the various object changes, indicating the simulated values. This scenario starts at 11 PM. She then modifies the "time" contextual value to "fast forward" the environment's state near her wake-up time. Reaching 6:30 AM, she notices that the description of the coffee brewer automation is placed over the related machine, but the panel is opaque, indicating a rule not in execution. At the same time, a dotted red connection line appears between the visualization and another panel placed over the smart plug to which it is connected. The panel over the plug is instead bright, indicating a rule currently in execution, which function is to deactivate that plug at nighttime, between 12 PM and 7 AM, and it is hence preventing the coffee automation to start.

To better assess how to define explanations with respect to the framework, we start with the **"When"** aspect. In this scenario the user is actively using the simulator to better understand what causes or prevents the activation of automations and possibly clarify her confusion, hence explanations are automatically triggered. For the **"What"** part, the main system goal, in this case, is error management since the simulator aims to help users to better collaborate with the system and to calibrate their expectations of the system's capability and functioning. From the intersection of the related table in the framework, the most suited explanation types are Input/Output, Why/Why-not, How, and Certainty. Concerning the detail level, priority should be given to a concise explanation of the why (in this case, making clear that the smart plug automation is preventing the coffee maker). Further details and explanation types can be provided on request, for instance, by giving concrete examples of contexts in which the specific automation can be activated. Concerning the **"How"** part, the modality is visual, using a mix of implicit (the red dotted line connecting the objects) and explicit (the opaque or bright panels with textual explanation) patterns.

4. Conclusions and Future Work

In this paper, we presented ARACS, an AR platform to define IoT automations with recommendation support. The platform provides users with dynamic suggestions on how to complete the automation they are editing based on the specific user, context, and textual representation of the context. The platform will include a simulator to allow users to load context snapshots and simulate the execution of the rules and the effects between them. We then introduced XAIR, a framework that aggregates the main XAI factors from the literature and allows for a systematic definition of the When, What, and

How of an AI explanation. Finally, we discussed how the intelligence aspects in the ARACS platform (recommender system and automations simulator) could be made explainable by applying the XAIR framework. For future work, we plan to refine the application's simulation part and introduce further features. An example can be the automatic evolution of the context snapshots, for instance, based on predictions from stored context data and knowledge of the environment [2] and also considering the effects of the activation of automations. We are also planning to extend the recommendation architecture to consider further data such as user profiles and assess the solution with different TAP rules datasets. We will also conduct user studies to assess the impact of introducing explanations into the platform.

5. Acknowledgements

This work has been supported by the PRIN 2017 “EMPATHY: Empowering People in Dealing with Internet of Things Ecosystems”, https://www.empathy-project.eu/.

6. References

[1] Chen, Xuyang, Xiaolu Zhang, Michael Elliot, Xiaoyin Wang, and Feng Wang. "Fix the leaking tap: A survey of Trigger-Action Programming (TAP) security issues, detection techniques and solutions." *Computers & Security* (2022): 102812.

[2] Coppers, Sven, Davy Vanacken, and Kris Luyten. "Fortniot: Intelligible predictions to improve user understanding of smart home behavior." *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 4, no. 4 (2020): 1-24.

[3] Corno, Fulvio, Luigi De Russis, and Alberto Monge Roffarello. "RecRules: recommending IF-THEN rules for end-user development." *ACM Transactions on Intelligent Systems and Technology (TIST)* 10, no. 5 (2019): 1-27.

[4] Corno, Fulvio, Luigi De Russis, and Alberto Monge Roffarello. "Empowering end users in debugging trigger-action rules." In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, pp. 1-13. 2019.

[5] Corno, Fulvio, Luigi De Russis, and Alberto Monge Roffarello. "HeyTAP: Bridging the Gaps Between Users' Needs and Technology in IF-THEN Rules via Conversation." In *Proceedings of the International Conference on Advanced Visual Interfaces*, pp. 1-9. 2020.

[6] Coutaz, Joëlle, and James L. Crowley. "A first-person experience with end-user development for smart homes." *IEEE Pervasive Computing* 15, no. 2 (2016): 26-39.

[7] Devlin, Jacob, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. "Bert: Pre-training of deep bidirectional transformers for language understanding." *arXiv preprint arXiv:1810.04805* (2018).

[8] He, Xiangnan, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. "Neural collaborative filtering." In *Proceedings of the 26th international conference on world wide web*, pp. 173-182. 2017.

[9] Huang, Bing, Hai Dong, and Athman Bouguettaya. "Conflict detection in iot-based smart homes." In *2021 IEEE International Conference on Web Services (ICWS)*, pp. 303-313. IEEE, 2021.

[10] Kulesza, Todd, Margaret Burnett, Weng-Keen Wong, and Simone Stumpf. "Principles of explanatory debugging to personalize interactive machine learning." In *Proceedings of the 20th international conference on intelligent user interfaces*, pp. 126-137. 2015.

[11] Liao, Q. Vera, Daniel Gruen, and Sarah Miller. "Questioning the AI: informing design practices for explainable AI user experiences." In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, pp. 1-15. 2020.

[12] Manca, Marco, Fabio Paternò, Carmen Santoro, and Luca Corcella. "Supporting end-user debugging of trigger-action rules for IoT applications." *International Journal of Human-Computer Studies* 123 (2019): 56-69.

[13] Wu, Q., Shen, B., Chen, Y. (2020). Learning to Recommend Trigger-Action Rules for End-User Development. In: Ben Sassi, S., Ducasse, S., Mili, H. (eds) Reuse in Emerging Software Engineering Practices. ICSR 2020. Lecture Notes in Computer Science(), vol 12541. Springer, Cham. https://doi.org/10.1007/978-3-030-64694-3_12

[14] Xu, Xuhai, Anna Yu, Tanya R. Jonker, Kashyap Todi, Feiyu Lu, Xun Qian, João Marcelo Evangelista Belo et al. "XAIR: A Framework of Explainable AI in Augmented Reality." In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, pp. 1-30. 2023.

[15] Yusuf, Imam Nur Bani, Lingxiao Jiang, and David Lo. "Accurate generation of trigger-action programs with domain-adapted sequence-to-sequence learning." In *Proceedings of the 30th IEEE/ACM International Conference on Program Comprehension*, pp. 99-110. 2022.

[16] Zhang, Lefan, Weijia He, Jesse Martinez, Noah Brackenbury, Shan Lu, and Blase Ur. "AutoTap: Synthesizing and repairing trigger-action programs using LTL properties." In 2019 IEEE/ACM 41st international conference on software engineering (ICSE), pp. 281-291. IEEE, 2019.