



End-User Development

Fabio Paternò

Contents

Introduction	2
Historical Evolution	3
EUD at the IoT and AI Time	7
Trigger-Action Programming	9
Automation Composition Paradigms	12
Real-World Deployment, Execution, and Monitoring	17
Recommender Systems in EUD for IoT Settings	19
Explainable End-User Automation Debugging	20
Augmented Reality-Based Support for Automation Control	22
Conclusion	23
References	24

Abstract

This chapter provides a description of the main motivations for the research work carried out in end-user development and discusses the state of the art in this area. It introduces the historical evolution of the field and then focuses on the various possible approaches to address the main recent technological trends (Internet of Things and Artificial Intelligence). For this purpose, it describes trigger-action programming, the possible approaches to end-user automation composition, experiences in real-world deployment, the possible role of recommender systems, and explainable end-user automation debugging. Lastly, current challenges and new frontiers are discussed.

F. Paternò (✉)
CNR-ISTI, HIIS Laboratory, Pisa, Italy
e-mail: fabio.paterno@isti.cnr.it

Keywords

End-user development · No-code · End-user software engineering

Introduction

The need for allowing end users to obtain digital systems that are not only easy to use but also easy to modify in terms of their user interfaces, behavior and functionalities emerged very soon. Such need has increased more and more with the penetration of digital services in all areas of our daily life. Each person can have different needs and preferences, which can also change over time, and thus would like to modify the available applications to better meet such dynamic needs, even if they are not a professional developer. End-user development (EUD) helps to solve this problem. It has been defined as “a set of methods, techniques and tools that allow users of software systems, who are acting as non-professional software developers, at some point to create, modify, or extend a software artefact” (Lieberman et al. 2006).

Historically, first the End-User Programming (EUP) (Nardi 1993) concept was proposed, but this concept is more limited than EUD, since EUD methods, techniques, and tools span the entire software development life cycle, including modifying and extending software, not just the programming phase. Some authors (Ko et al. 2011) proposed the End-User Software Engineering (EUSE) concept as well, with the aim of finding ways to incorporate general software engineering activities into users’ existing workflow, with an emphasis on the quality of the software end users create, modify, or extend. EUSE takes a different perspective compared to EUP and EUD because it focuses on systematic and disciplined activities carried out throughout the system life cycle to guarantee the quality of the code created by end users. A literature review of the contributions in such areas in the period January 2000 to May 2017 is presented in (Barricelli et al. 2019), which grouped them according to seven dimensions: type of approach, interaction technique, phase in which the approach is adopted, application domain, target use, class of users, and type of evaluation.

Over time, a community has consolidated in the EUD area. Often there are papers in main human-computer interaction or software engineering conferences that address such topics, and there is also a dedicated biannual symposium (IS-EUD (<https://iseud.net/>)). Several approaches have been put forward in this area, and in general they evolved also considering the more general evolution of the digital technological trends. One relevant aspect of EUD approaches is the type of representations and metaphors they adopt with the aim to represent the development concepts to people without programming experience (Paternó and Santoro 2019). Thus, they should use concepts and representations that are used in the users’ real world, with the goal to be more immediately understandable. In this way, metaphors provide users with easily understandable cognitive hints expected to facilitate the creation or customization of an application by decreasing the learning effort needed by a nonprofessional developer to manipulate programming concepts and artifacts.

One example is the puzzle metaphor, which has long been considered for various types of EUD solutions where the shape, size, and colors of the puzzle pieces are used to provide visual cues about how to compose the relevant elements. One of the first tools adopting this metaphor was Scratch (Resnick et al. 2009) for supporting children in creating multimedia applications, then it has been used for supporting development of mobile apps; see AppInventor or (Danado and Paternò 2012), and more recently for creating automations (Mattioli and Paternò 2021). One distinct but often connected aspect is the programming style offered by the considered EUD approach. For programming style, we consider how the EUD environment allows the creation or modification of an application. Thus, in this case we focus on the concrete solution in terms of programming. Examples of programming styles are programming by example, trigger-action-based approaches, natural language techniques, mashups, and mock-up-based and tangible programming techniques.

In general, EUD has been considered in a wide variety of application domains including business and data management, education and teaching, games and entertainment, and healthcare and wellness, as well as technologies such as mobile apps, robotics, smart objects and environments, and Web applications and mashups.

The goal of this chapter is to allow readers to gain knowledge and skills in addressing problems in end-user understanding, creation, control, monitoring, and debugging of interactive applications of current technologies. It reviews the state of the art in terms of concepts, techniques, languages, and tools in order to allow readers to understand the design space of the possible solutions and associated issues so as to better apply them and think about new ones.

In the chapter, after describing the historical evolution of the field, we analyze the main current technological trends and how they have impacted the EUD domain. We then focus on trigger-action programming, which is particularly relevant in the context of applications that exploit Internet of Things (IoT) and Artificial Intelligence (AI) technologies, which are currently the most widespread and adopted technologies. Thus, we discuss various approaches for composing automations in such settings, and we introduce the issues and potentialities when deploying them in real-world scenarios. The next sessions are dedicated to discussing how to exploit recommendation systems in such environments, the support for explainable debugging, and the possibilities introduced by augmented reality in terms of end-user development activities. We conclude with some final remarks and indications of future challenges.

Historical Evolution

Historically, it is possible to identify various generations of EUD approaches. In the evolution of this area, the start of a new generation has been characterized by the advent of some mainstream ICT technology, which raised the need for new application types or allowed users to exploit new platforms to obtain their applications.

The first approaches relevant to EUD were put forward in the late 1980s, soon after the advent of graphical desktop systems, which made it possible to support the first emerging end-user development needs. Over time, several technologies had an effect on the possibilities for EUD, including language technologies and social systems. Among them, the rapid increase in the use of the Web, with its open interfaces (as opposed to offline desktop applications) to support computational work, made possible the design of a number of interactive tools to support EUD. The third generation corresponded to the success of touch-based mobile devices, characterized by sufficient interaction and computation resources to directly support development activities. The last generation aims to empower users in order to exploit the continuously increasing availability of smart things and robots, two types of technologies that share the use of various types of sensors and actuators.

In the late 1980s and early 1990s, the first contributions, which mainly addressed how to improve end-user programming (e.g., Myers and Buxton 1986; Nardi 1993, Cypher 1993), were put forward. Then, on the European side, there was the Network of Excellence on EUD (2002–2003) (<http://hiis.isti.cnr.it/projects/eud-net.htm>), which was useful for creating a community with an open view of how the user can be empowered in the development cycle. In parallel, the NSF started a project (EUSES, 2003–2004) on end-user software engineering, which stimulated a more systematic view on how to approach the various software engineering phases from the user side. Such research efforts stimulated an agreed definition of EUD (Lieberman et al. 2006) and various research initiatives, for example, the manifesto for metadesign (Fischer et al. 2004). Regarding the exploitation of Web technologies in this perspective, a book (Cypher et al. 2010), which was a follow-up of a CHI workshop, provides a good overview of various approaches in this area. The first attempts to exploit mobile technologies in this area were put forward around 2008 (e.g., Carmien and Fischer 2008; AppInventor 2010). An overview of initial approaches in the area of EUD for IoT applications appeared in the dedicated TOCHI special issue (Markopoulos et al. 2017).

In the first generation, corresponding to the wide availability of graphical desktop systems, the contributions mainly focused on two types of approaches: those based on spreadsheets and those based on visual languages. A long-standing example of domain expert development of interactive applications is the spreadsheet application: Laymen as well as professionals can develop or adapt computational models useful for accounts, planning, etc. with a minimal overhead of learning programming concepts and conventions. To reach such a broad audience, the EUD aspects should be almost transparent to users, e.g., spreadsheet users may not even be conscious of the fact they are programming when processing data. At least, EUD should present a very low threshold to get started, while letting users progress far in the value and even complexity of the software they create. In the case of spreadsheets, the interest started with VisiCalc, then continued with Lotus 1–2–3 and Excel, and the focus has been on facilitating the definition of expressions to apply to multiple cells, and then testing their results, and the analysis of the relationships among the values of the cells involved.

In the case of visual languages, one often used approach is based on icons associated with high-level functionalities, which are connected through arrows to indicate the data that can flow across them. For example, LabVIEW (<https://www.ni.com/it-it/shop/labview.html>) is an environment for creating circuit simulations and other analysis programs. Each box represents a computational component, while lines indicate the flow of data. Usually, in this type of approach there is some support to check whether the output from one functionality is actually compatible with the input of another one. One further visual approach that soon raised interest is based on the jigsaw metaphor. The idea is to visually represent the jigsaw puzzle pieces so that each element corresponds to a function or a programming element, and their shapes provide hints about how many connections they can manage both for receiving input and producing output. Scratch (Resnick et al. 2009) was one of the first environments to adopt this type of representation. In visual languages, one common design aspect has been how abstractions are used to hide the implementation details (Paternò 2013). In some cases, the main visual elements have been associated with high-level functionalities developed by programmers, so that end users need only to compose them without knowing how they were implemented. In other cases, the visual elements correspond to low-level programming constructs, and the end users should specify how the desired interactive program should behave at a more detailed level. In that period, another approach to EUD that emerged was programming-by-example, in which the user demonstrates an example of what the program should do, from which the programming environment infers a more general specification supporting the desired behavior. Peridot (Myers and Buxton 1986) was one of the first examples applying this approach to creating interactive components.

Over time, the Web has become the most common user interface because it can be accessed through most devices and provides an open interface (the Document Object Model), which can be exploited to interactively manipulate Web applications, even by people other than the original developers. The programming-by-example approach has been implemented in Web environments through different mechanisms. Nichols and Lau (2008) describe a system that allows users to create a mobile version of a Web site through a combination of navigating through the desired portion of the site and explicitly selecting content. Macías and Paternò (2008) take a similar approach, in which users directly modify the Web page source code. These modifications are used as a specification of preferences, which are then generalized and applied to other pages on the same site through the support of model-based user interface description languages. CoScripter (Leshed et al. 2008) is a system that allows users to record, share, and automate tasks to perform on the Web, and provides a repository where the scripts created are shared. In CoScripter, scripts are recorded as natural language commands that can be modified by the user without having to understand a programming language. In this area, one approach often considered is the mashup approach characterized by the possibility of creating new applications by interactively composing components from existing applications. NaturalMash (Aghaee and Pautasso 2014) is a Web-based environment that allows nonprogrammers to exploit existing Web resources by combining their input/output. NaturalMash users start defining a mashup by picking ingredients from a toolbar

that includes services/contents available through Web APIs. Then, users specify how to bind components together through a natural language subset. However, the mashup components associated with textual expressions are predefined and require preprocessing by expert programmers. PEUDOM (Matera et al. 2013) is another Web-based platform that allows end users to compose components associated with registered Web services into a mashup. Components are defined by professional developers and can subsequently be connected by means of drag-and-drop actions and by selecting the binding properties from some dropdown menus. Ghiani et al. (2016) put forward a graphical environment in which users create new mashups by directly selecting interaction elements, content, and functionalities from existing Web applications without requiring the intervention of expert developers. Then, users just need to exploit a copy-paste metaphor to indicate how to compose the selected interactive content and functionalities in the new mashup.

A different perspective on exploiting Web technologies that can be relevant for EUD has been introduced by Webstrates (Klokmoose et al. 2015). It is an environment for creating shareable dynamic media that blurs the distinction between documents and applications, showing how software can become reprogrammable and extensible in a collaborative fashion.

The advent of touch-based mobile devices in the mass market enabled new opportunities for EUD. On the one hand, the mobile device can be the platform through which users can create or customize their applications, and on the other hand, applications have to be able to consider that the surrounding context of use is no longer fixed, and so they have to adapt to changes that can dynamically occur in the environment. Carmien and Fischer (2008) describe a framework for customizing mobile applications to help people with cognitive disabilities. A graphic editor, intended to be used by the caretakers, facilitates the management of the task-support scripts for helping the disabled. Ghiani et al. (2009) developed an environment that allows customization of mobile solutions for museum guides, and it also allows the generation of application versions for stationary systems with large screens. Puzzle (Danado and Paternò 2012) supports editing on a touch-based smartphone by using the jigsaw metaphor to convey the concepts of connecting high-level functionalities, and a solution, inspired by the work by Cuccurullo et al. (2011) using the colors to indicate the associated data types, thus providing intuitive cues to help the users to correctly connect the outputs and inputs of the jigsaw pieces. App Inventor (2010) addresses the application development at a more detailed granularity, asking the end-user developers to use jigsaw pieces representing low-level programming constructs and specify what should be done when low-level events occur.

In the last decade, the most important technological trends in the digital field have been the advent of the Internet of Things, characterized by the explosion of available connected sensors and objects, and artificial intelligence, which has introduced several automations based on the analysis of large amounts of data. The next section discusses how such technological trends have impacted in the EUD field.

EUD at the IoT and AI Time

If we look at the current technological trends, we can notice that there is an explosion of connected objects and sensors, and current forecasts indicate 30.9 million IoT objects available in 2025, while for devices such as smartphones, laptops, and desktop computers the forecast indicates a slight increase, rather small in percentage (<https://www.statista.com/statistics/1101442/iot-number-of-connected-devices-worldwide/>). Indeed, for example, recent statistics suggest that there are currently over 258 million “Smart Homes” worldwide, and 40% of US households have adopted some Smart Home technology.

One consequence of such a technological trend along with the introduction of artificial intelligence is that in many daily environments we encounter more and more automations. Examples of daily automations are:

- In a store, automations for sending personalized ads to customers’ smartphones based on the movements and interactions with store products.
- In an older adult residence, automations for sending personalized messages to the older adult to promote healthy behavior, for example, a warning on the preferred device (e.g., TV) to encourage them to exercise more, based on the activities detected in the morning.
- In an industrial site, automations to activate safety alerts to workers on their smartwatch based on the dynamic presence of some potentially dangerous equipment.
- In a smart home, automations to better control the activities according to user preferences, routines, devices, and context, e.g., if it is a weekday, from 9 to 17, and a sensor detects a certain movement, the application should flash red lights in the smartphone, with an image from the home webcam.

Generally, how people interact with intelligent spaces can be classified into three categories (van Berkel et al. 2021):

- Intermittent interaction, in which users provide inputs and the AI system generates suggestions accordingly. For example, the user asks for the weather forecast for a given town the day after.
- Continuous interaction, where the user continuously provides inputs and the system monitors them, and intervenes when necessary or relevant. For example, automatic support for car driving.
- Proactive interaction, which occurs when the intelligent system monitors the context of use, and at a certain state it activates some functionalities without explicit user request. For example, a home automation system could increase the brightness of the surrounding lights when detecting both user presence and decreasing brightness levels.

Such intelligent environments are penetrating many areas of our daily life, but sometimes the user experience is not satisfactory. For example, Yang and Newman (2013) report on a study in which the NEST thermostat was deployed in some homes and then 19 participants reported on their experience. They used it for at least 1 month and were interviewed at different points. Some users reported comments such as the system fails to learn the user intent (e.g., “it is not adaptive enough to my recent needs”) or its behavior is difficult to understand (e.g. “I did not understand how much data was necessary to obtain the desired behaviour”).

In general, we can notice that the combination of Internet of Things and Artificial Intelligence provides great opportunities, and new possibilities, but there are also risks and new problems: There can be intelligent services that eventually generate actions that do not match the real user needs; the introduced automations can generate unwanted effects; people may have difficulties in understanding the automatically generated automations.

Thus, one key challenge is how to provide tools that allow users to control and configure smart environments consisting of hundreds of interconnected devices, objects, and appliances. Tools that allow people to obtain something I like to call “*humanations*,” which are automations that users can understand and modify.

In this respect, the solution lies in finding the most suitable trade-offs between artificial intelligence (which aims to provide algorithms and techniques to generate automations in order to satisfy users’ needs) and end-user development (which focuses on tools and methods to empower humans to control the automations to meet their needs).

Generally speaking, the possible ways to obtain automations are described in Fig. 1. The pure AI approach starts from an analysis of the requirements based on the user needs and the data available. Once the relevant data are collected, a model for their analysis is developed and validated for obtaining the automations, which are then deployed and executed with some run-time support. Such intelligence can also be exploited to recommend automations to the end users, who can then decide which ones should be made active. EUD tools usually follow a different approach where users can directly create the relevant automations. In this work, they can also be supported by some recommender system, which can make their suggestions either from the analysis of the user behavior or from the analysis of some automation repositories. In any case, such different paths in the creation of automations are not mutually exclusive, and in general we expect that in human-centered intelligent environments they should coexist.

Whatever approach has been used to obtain the automations, a key point for their acceptance is their transparency. Various aspects are relevant for obtaining a transparent solution:

- The level of user control
- Whether it is clear the granularity of the set of objects involved in the considered automations
- The temporal aspects of automations concerning the triggers and the actions
- The availability of analytics in terms of data indicating how they have behaved

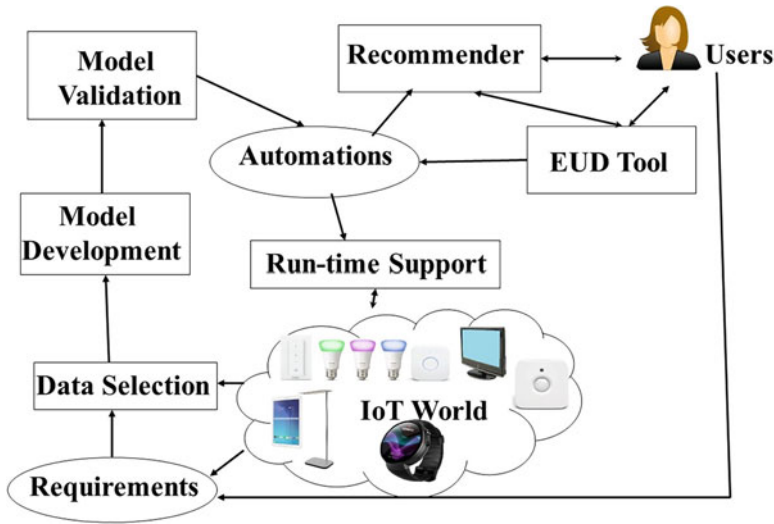


Fig. 1 Reference model for daily automations

In particular, the control levels are:

- Perception – do users perceive that an automation is working?
- Understanding – do users know why, when, and how the automation generates some effect?
- Predictability – are users able to predict what automations will take place in the future?
- Modifying – are users able to change something in the automation to better meet their specific needs?

Trigger-Action Programming

Trigger-action programming (Ur et al. 2014) has often been adopted for representing, managing, and executing automations. It is a suitable approach for EUD since it does not require particular algorithmic ability, and is intuitive since it requires indicating the relevant events or conditions and then the desired actions.

Triggers represent situations that are relevant to the users and may relate to their emotional or physical state, or the surrounding environment or available devices or applications. Trigger information is derived from various sensors (for example, movement, proximity, light, noise, breathing, heartbeat...) and applications and services.

Actions represent what the objects, devices, and applications available are able to do. They allow control of objects (such as turning on/off lights, opening/closing

doors, and activating TV/radio), sending reminders or alarms, and performing user interface changes.

Triggers can be composed of events and/or conditions. Sometimes it is not immediate to distinguish between events and conditions. Failure to do so can cause unwanted behavior, e.g., opening a door at the wrong time or turning on the heater when it is not needed. For example, these two rules have different effects:

- When I get home, the bell rings.
- If I am at home, the bell rings.

And the latter can be rather annoying. To better understand the difference between events and conditions, it is important to remember that an event happens at a point in time, for example, when the user enters a room, when it starts to rain, when the kitchen temperature exceeds 30 degrees, at 8 o'clock, while a condition is a state that lasts for a longer period of time, for example, while the user is inside a room, as long as it is raining, if the kitchen temperature is over 30 degrees, and between 8:00 and 11:30. The use of different keywords helps to differentiate them.

There are also other useful trigger operators: rules that are triggered if an event does not occur in a specific interval of time (e.g., “When the medicine has not been taken between 10 a.m. and 11 a.m.”); rules that are triggered when a specific ordered sequence of events occurs (e.g., “When the user enters a room and then exits” or “When the temperature becomes higher than 20 degrees and then the humidity level becomes higher than 50%”); and when an event occurs a specified number of times (event iteration) (e.g., “When the user goes to the bathroom 5 times during the night”).

Figure 2 shows the possible types of trigger composition when they are concatenated through a logical AND. The composition of an event and a condition is an event that occurs while the condition is true. The composition of two events is something that is unlikely to happen since both are instantaneous and it is very difficult that it can occur at the same time. The composition between two conditions is still a condition that is verified in the interval given by the intersection of the two corresponding interval times.

It is important to avoid inverting the triggers because the resulting automation can have different effects. If we consider:

- R1: WHEN it starts raining, IF the garage door is open DO close the garage door.
 R2: WHEN the garage door opens, IF it is raining DO close the garage door.

In one case, the garage gets wet: Indeed if we consider the case when first the door opens (an event that leads to the condition “garage open”) and then it starts raining, with R1 the door is closed as soon as it starts raining, while in R2 nothing happens because it considers only when the garage door opens while it is raining, and since in the example the garage door opens before while it is not raining, then nothing happens. However, both R1 and R2 are necessary, since R2 prevents the garage from getting wet if someone opens it while it is already raining.

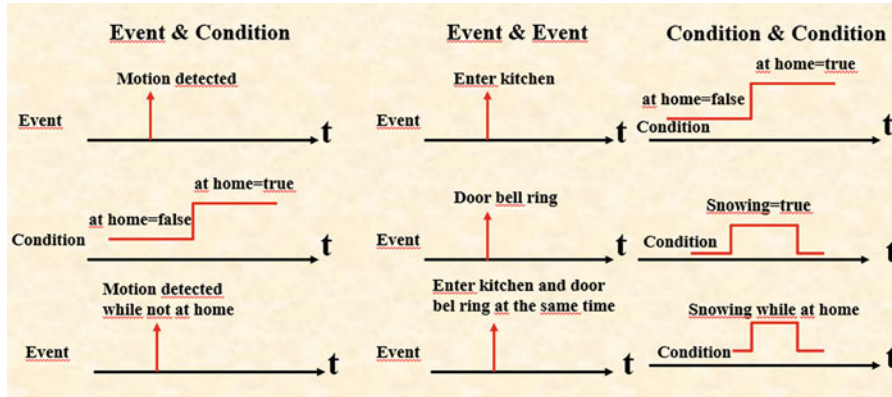


Fig. 2 Possible types of trigger composition

Also actions can be categorized according to their temporal duration (Huang and Cakmak 2015): instantaneous (e.g., sending an email); extended (e.g., brewing coffee); and sustained (e.g., turning the light on).

When composing triggers and action, some attention should be paid to their temporal aspects and how they can be related. In particular, when the trigger is a condition, then it can be useful to analyze the type of action. In the case of sustained action, users expect that when the condition is no longer true then the sustained action should be stopped. For example, in the case of the rule: *IF the time is between 8 and 11:30 p.m., DO turn on the living room light*, users expect that the living room light will be OFF otherwise (i.e., when the time is before 8:00 p.m. OR the time is after 11:30 p.m.).

The case of a condition composed with an instantaneous action (e.g., *IF I am at home, DO send email to John*) can be problematic since it is unclear whether the action should be performed once or multiple times as long as the condition is true. Thus, if the action should be performed once, then an event should be used in the trigger side to indicate when it should be performed.

More generally, Brackenbury et al. (2019) have tried to categorize the various possible TAP bugs. One group is dedicated to controlling flow bugs, such as infinite loops among automations or repeat triggering, which occurs when composing a condition with an instantaneous action. Another group is dedicated to timing bugs (such as rules failing to account for the extended timing of an action, e.g., brewing coffee).

One further aspect to consider is that there can be several possible rule-writing styles depending on the perspective from which the automations are considered. In particular, it is possible to identify the following styles (Corno et al. 2021):

- Device-centric, a rule whose subject is the physical medium with which it is executed, e.g., “when the kitchen motion sensor becomes activated”

- Information-centric, a rule whose subject is the underlying information, regardless of the physical medium with which it is manipulated, e.g., “when I receive a Facebook notification”
- People-centric, a rule where users, their actions, and/or feelings are at the center of the interaction, independently of any physical and virtual medium, e.g., “when I enter the kitchen”

Automation Composition Paradigms

Several paradigms have been investigated for automation composition. A composition paradigm is characterized by how it presents the relevant concepts and interacts with users, and how it supports the rule development process. In the area of automation composition, we can identify at least the following paradigms (Gallo et al. 2021):

- Data flow, representing how information goes through the various components
- Visual wizards, aiming to drive users by limiting their possible selections
- Block-based, using the puzzle metaphor to suggest possible compositions
- Conversational, exploiting natural language and AI to get user input, process it, and request clarifications

In data-flow representations, the basic idea is to have nodes corresponding to processing elements and the arrows indicate how the information should flow: which elements should receive the results of the various processings. Figure 3 shows an example of tool supporting the composition through data-flow representations. It is Node Red (<https://nodered.org/>) a tool that provides sets of nodes and flows, with the aim of supporting low-code programming for event-driven applications. When a node is selected, it is possible to edit the corresponding scripts. This tool aims to facilitate the development of automations but still requires some programming knowledge to indicate and understand the various scripts corresponding to the nodes.

The visual wizard is probably the most adopted compositional style for automations. Currently, the most widely used tool in this area is IFTTT (<https://ifttt.com/create>). It is a Web and mobile environment that allows users to create and share rules, called “applets,” in the form “if trigger then action.” Triggers and actions can be chosen from existing services (for example, Facebook, Evernote, Weather, Dropbox, etc.). In 2017, it was estimated that 320,000 applets involving 400 service providers were installed more than 20 million times. Services are peripherals, hubs, wearable devices, devices in the car, online services, social networks, messaging. Figure 4 shows some examples of automations created with IFTTT. When creating a rule, the tool requires to indicate the channel (e.g., Facebook or Google Drive) for both the trigger and the action. The tool has some limitations: It does not distinguish between events and conditions, it is not easy to extend the list of the connected applications, it shows long lists of potential channels to compose where it is easy to get lost. In September 2020, it switched to a limited freemium model with a

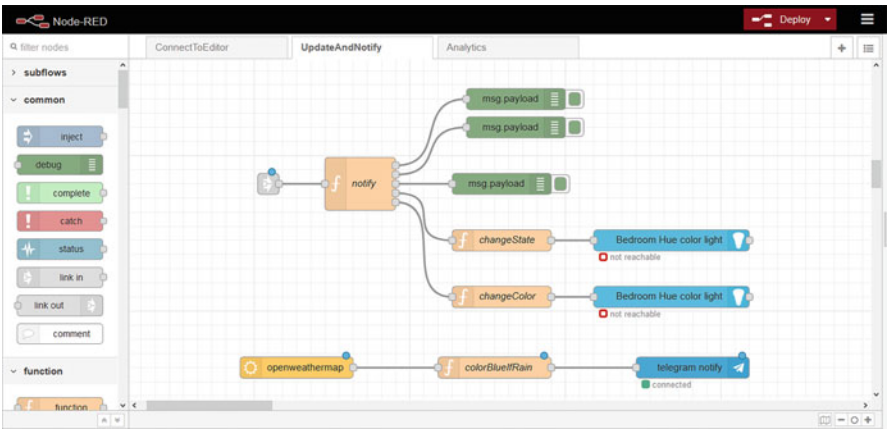


Fig. 3 Example of data-flow tool

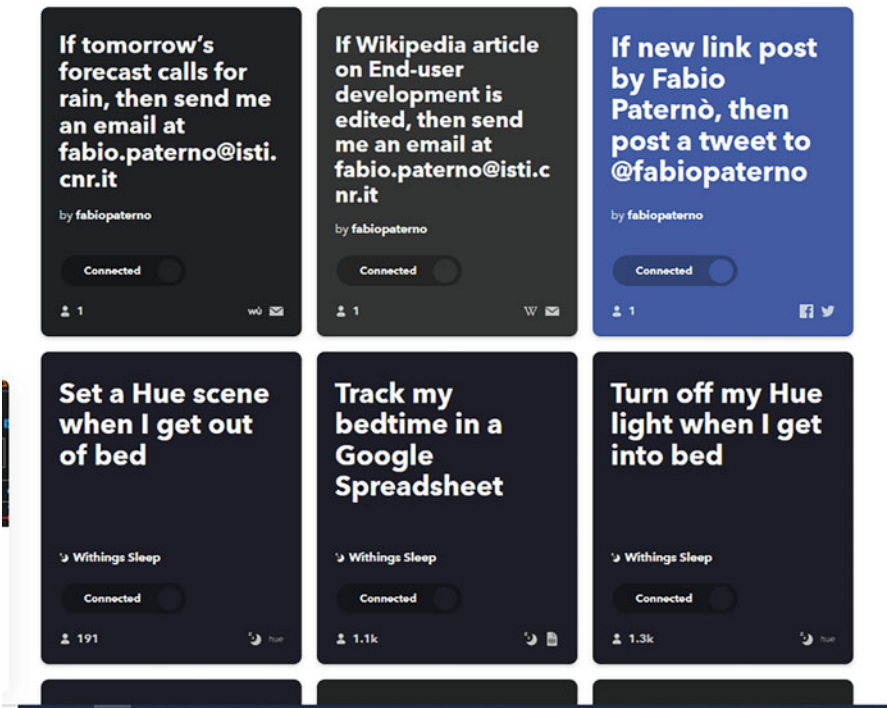


Fig. 4 Example of automations created with IFTTT

subscription-based version (“IFTTT Pro”), which allows rules to use conditional statements, multiple actions, and query data for more complex tasks, while users were limited to three custom applets in the free version.

One tool aiming to overcome some of the IFTTT limitations still adopting the visual wizard composition style is TARE (<https://tare.isti.cnr.it/RuleEditor/login>) (Trigger Action Rule Editor). It supports the creation of automations with multiple triggers and actions. When creating a trigger, it requests different keywords for explicitly differentiating events and conditions. Triggers are logically organized according to the contextual dimensions they refer to (user, environment, and devices).

As Fig. 5 shows, the user interface has a main area where the user can select the relevant trigger or action. On the left, there is a sidebar, which indicates the process for completing a rule, what the user has performed, and the next possible steps. On the top, there is a natural language description of the rule under composition.

The block-based composition paradigm has been often considered in the EUD area, e.g., Danado and Paternò (2012)) because several designers consider it potentially more creative, easy, satisfying, and engaging. It has been used in several domains, such as education, IoT, and industrial robotics. There are libraries supporting its application, such as Google Blockly. An example of the block-based composition paradigm is BlockComposer (<https://giove.isti.cnr.it/demo/pat/>) (Fig. 6). The user can select the relevant element for the rule under creation on the left and then drag it in the main area where the composition of the elements is performed by plug-and-play the various puzzle pieces. Their color and shape have been designed in such a way to facilitate their management in the composition process (e.g., there are different colors for triggers and actions).

More recently, given the increasing evolution of the natural language processing technologies and the support of artificial intelligence, one further compositional style for automations has been made possible, the conversation style in which the idea is to support a conversation between the user and the system in a way similar to the conversation between humans.

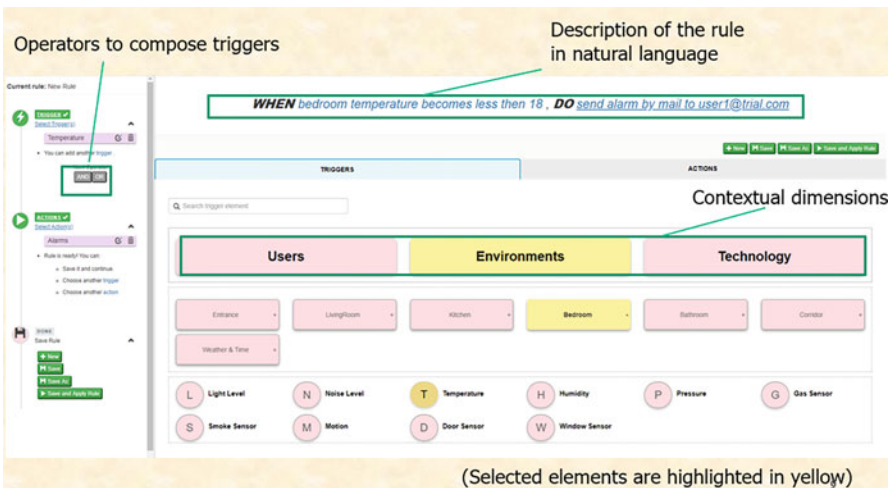


Fig. 5 The TARE user interface

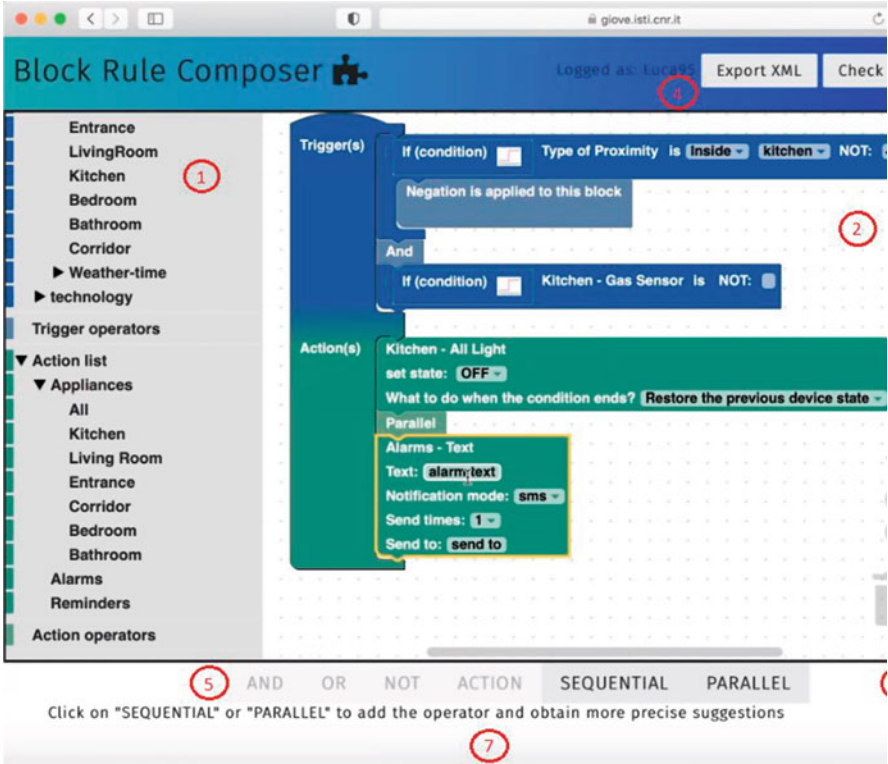


Fig. 6 The BlockComposer user interface

One example of conversational agent in this area is RuleBot (Gallo and Paternò 2022), a chatbot implemented with Dialogflow support for allowing users to create automations (Fig. 7). The solution aims at avoiding making the conversation fragmented and unnatural. After chatbot welcome, the user can enter the desired automation with any number of triggers and actions. Intents have been created for each possible trigger and action, and associated with some training phrases and entities. When one intent is recognized, the chatbot checks the presence of the necessary entities and asks for the missing information. In the conversations, users can delete the last item entered, and ask for the description of the rule so far created.

A study comparing different compositional styles is reported in (Gallo et al. 2021). As represented in Fig. 8, it emerged that the visual wizard can be effective and efficient when creating simple rules since the editor tends to provide indications for what steps to perform, but when the rules increase in complexity, then the wizard can become tedious and long. The chatbot can have a complementary effect since at the beginning, even with simple rules, there can be some errors since users may not immediately understand how it expects to receive requests. However, once the user and the agent tune their communication languages, the interaction becomes fluent, even with rules more complex.

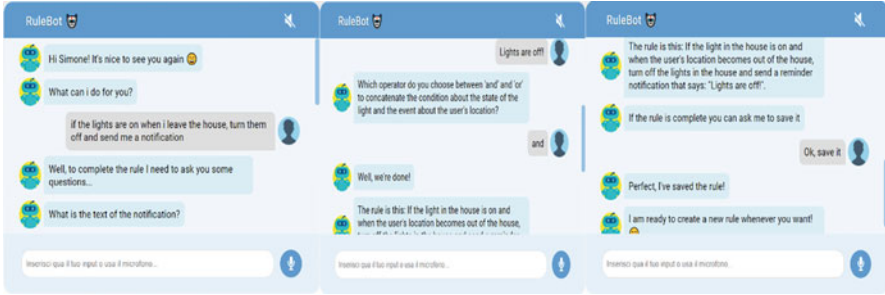


Fig. 7 An example of conversation with RuleBot

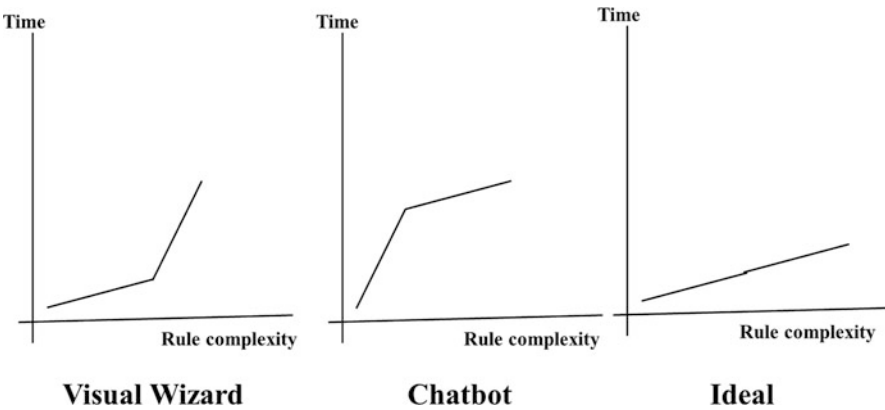


Fig. 8 Usability curves in automation creation

More generally, recent advances in Natural Language Processing (NLP) have greatly improved conversational capabilities and language-related downstream tasks through Large Language Models (LLMs), such as GPT-3 and ChatGPT. With the use of prompt engineering techniques, they can perform zero-shot and few-shot tasks from a small set of examples, without the need for training or fine-tuning a new model. However, applying these models to real-world problems requires further research since they exhibit probabilistic and not deterministic behavior so even a small change in the input may lead to the generation of an unexpected or inaccurate output. A new version of RuleBot (Gallo et al. 2023) aims to integrate the use of ChatGPT and Rasa to improve the version implemented in Dialogflow. In this case, the use of ChatGPT is focused on splitting complex rules specifications by the user into simple commands and managing possible breakdowns, while Rasa provides an transparent open source support for intent classification.

Real-World Deployment, Execution, and Monitoring

One important aspect to consider is how such environments to create, control, and modify automations can be deployed in real world for their actual use in practice in daily environments.

One first experience was reported in (Corcella et al. 2017). It concerns the deployment of a personalization platform prototype in a flat where three students live, and have used a supporting platform for about 1 month. Several sensors and connected objects were installed in the flat and connected with a rule editor in such a way that it was also possible to execute the created rules. The users could create their rules autonomously, but then in the execution the rules referred to the same rooms. This determined some interesting issues:

- Conflicts on shared resources, for example, two conflicting rules were created: (i) In the morning, the lights must be off; and (ii) WHEN the user arrives home, DO switch the lights on. Thus, what should it happen if the user arrives at home in the morning?
- Conflicts on people's habits, for example, one student created the rule WHEN the time is 8:30 a.m., DO turn on the radio, which was rather annoying for another student who liked to sleep at that time.

A more extended experimentation was carried out in the international AAL project PETAL (<http://www.aal-petal.eu/index.html>), whose goal was to support personalization for supporting older adults with mild cognitive impairments. Thus, the goal was to consider users with cognitive issues, e.g., tendency to forget tasks/events..., increased risk of social isolation, and depression. The platform aims at monitoring environment and user's behavior, and controlling applications and devices, and allows formal and informal caregivers to configure the functionalities of the technological support to control lights and other digital devices when relevant events occur, thus obtaining personalized control of lights and other digital appliances, personalized warning messages issued in risky situations, and persuasive messages to stimulate the elderly in healthier habits (e.g., do more physical activity).

The resulting platform is TAREME, a platform for specifying, executing, and monitoring personalization rules. Its high-level architecture is described in Fig. 9. Domain experts (in the case of the PETAL project, they were the caregivers) can create personalization rules. Such rules are sent to a Rule Manager, which determines when to execute them. For this purpose, it considers the information generated by the context manager, which is a server that receives information from the sensors, connected objects, and services, and indicates when an event occurs or a condition ends. In this way, the rule manager can detect when the triggers are verified and send the associated actions to perform to the relevant objects, devices, or applications. In addition, there is a monitoring component that is informed when rules are triggered, activated, or deactivated, and can provide information about their performance to the domain experts.

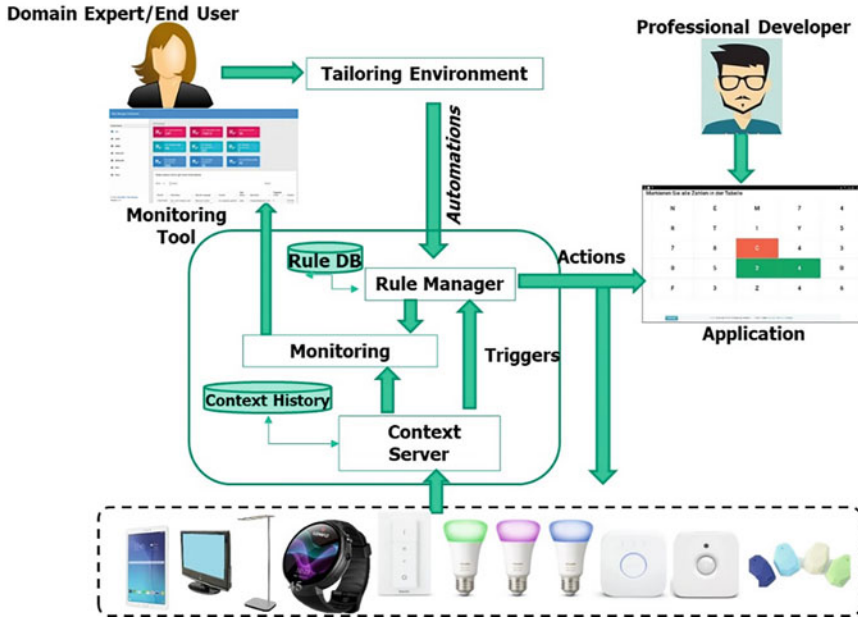


Fig. 9 The high-level architecture of the TAREME platform

Such platform has been deployed in 14 trials in various countries (Italy, Austria, and Romania) during the PETAL project with the goal to support older adults in structuring daily activities (e.g., to improve their sleep–wake rhythm), generate alarms or reminders through different channels (SMS/mail/app notification or light-based), support spatiotemporal orientation (e.g., find bathroom at night), improve visual comfort, encourage/improve healthy lifestyles, encourage social activity, improve emotional status, and support caregiver’s awareness.

Many automations were created to support different types of users’ goals, listed below with a corresponding example:

- **Safety.** WHEN the user exits home and IF the time is between 11 p.m. and 7 a.m., DO send a text message to the caregiver.
- **Comfort.** WHEN motion is detected in bedroom and IF the time is between 11 p.m. and 6 a.m., DO turn on lights to go to the bathroom.
- **Well-being.** IF the training time is less than 30 min and WHEN the time is 6 p.m., DO send a reminder.
- **Health.** WHEN NOT (taken medicine) between 08:00 and 09:00, DO send one alarm by text to caregiver.
- **Energy saving.** IF user is in bed AND the time is between 11:00 p.m. and 06:00 a.m., DO turn off all lights in the bedroom.
- **Socialization.** If tomorrow’s forecast is good AND tomorrow’s temperature is above 20 degrees Celsius, when it is 6 p.m. remind the user to call a friend to plan for a walk together tomorrow.

During the trials, the platform was able to support remote analysis and monitoring of the created automations and their actual use (Manca et al. 2021). Thus, it was able to provide information regarding the following: number of created rules, number of active rules, number of times rules have been triggered, the distribution of occurrence of trigger types in the set of created rules, distribution of occurrence of action types in the set of created rules, and time interval when the rules have been active.

Another study carried out in Finland (Salovaara et al. 2021) involved four small families, each including a technology expert, and focused on social needs that smart home toolkits should address. They found various types of examples such as utilitarian automations for elimination of repetitive tasks or notification about important things or hazard-preventions, warnings, or monitoring.

Different goals emerge when this type of personalization platform is considered for industrial settings. A study (Manca et al. 2022) carried out in a paper factories district has reported how in this context the most important goals for the considered automations are efficiency, safety of workers, quality of production, and predictive maintenance.

Recommender Systems in EUD for IoT Settings

Recommendations can be useful in various ways in this area. Generally speaking, we can identify two ways to exploit them in the rule creation process:

- Rule editing is comprised of multiple steps, which can benefit from recommendations that can be generated from some repositories of previously created rules.
- Recommendations can also be generated automatically taking into account actual user behavior for better support.

An example tool supporting the first type of recommendation use is Block-Composer, which was introduced in a previous section. The visual editor for creating rules is integrated with a recommendation system. The recommendations are shown in a pane on the right side. Two modes for providing recommendations are supported: step-by-step, in which at each step of the rule composition process suggestions for an additional piece of the rule appear, or full rule, in which the system suggests complete rules that can be used to replace the one currently being specified or to add pieces to integrate to it. In this way, beginners can benefit from being guided, while advanced users can discover new possibilities on their own. This solution exploits Compact Prediction Trees, which in general support consistent accuracy on different datasets, and are scalable and explainable. This type of model uses the elements of the previously considered sequences to obtain predictions on unknown subsequent sequences. For this purpose, it considers rules from a repository, deconstructs such rules into sequences of elements, and generates a neighborhood graph. Then, by traversing the graph it obtains a rule part that matches the user's currently inserted elements in the rule composition for generating

the recommendation. It exploits the data set of personalization rules available at https://github.com/andrematt/trigger_action_rules. There are also datasets of IFTTT rules (Ur et al. 2016).

In general, to identify relevant rules various criteria are possible. In collaborative filtering, if user A has specified a rule, which is also indicated by user B, then it is likely that other rules by user B are relevant for user A.

Recommendations obtained through generalization of the content of some part of the existing rules are possible. For example, if a rule says that when the user enters the bedroom then the lights should be on, one possible suggestion can be that when the user enters any room then the lights should be on. Another type of recommendation can be obtained by trigger refinement, which means narrowing the instances when the trigger should be fired by adding conditions that make the rules more suitable to meet needs more precisely specified.

Other recommendations can be based on the actual user behavior to detect their preferences. For example, they can be device-oriented: If the user prefers to use some specific device, then it can be meaningful to suggest rules that exploit it, for example, for sending alarms. One further aspect is to have location-dependent recommendations, which aim to provide suggestions when the users are in a specific area that they seem to prefer. There are also time-oriented recommendations, which are rules with triggers associated with specific periods of the day when the user is more inclined to receive information (e.g., reminders).

One example of a system providing recommendations based on the user's behavior is RuleSelector (Srinivasan et al. 2018). It monitors user behavior through the smartphone; thus, it identifies contexts of use (e.g., evening, home, connected to Wi-Fi, and watching TV), and associated actions. Then, according to some metrics, filters the set of rules that seem most relevant to suggest to the user. The metrics used are the following: confidence (the likelihood of the user performing an action in a given context), contextual specificity (how temporally precise the precondition contexts are), interval count (the number of unique instances when the rule occurred), and total action coverage (the proportion of instances of the rule that involves the considered actions with respect to the total).

Explainable End-User Automation Debugging

When several automations are active, various issues can occur, and there is a need to detect and explain them to users. Figure 10 shows three possible types of issues (Chen et al. 2022). One common issue is rule conflict, which occurs when different rules require performing conflicting actions at the same time. For example, if we consider the rules:

Rule 1: IF the time is after 9, DO open all the windows in the house.

Rule 2: IF the time is after 21, DO close all the windows in the house.

Rule 3: IF the weather condition is rainy, DO close all the windows in the house.

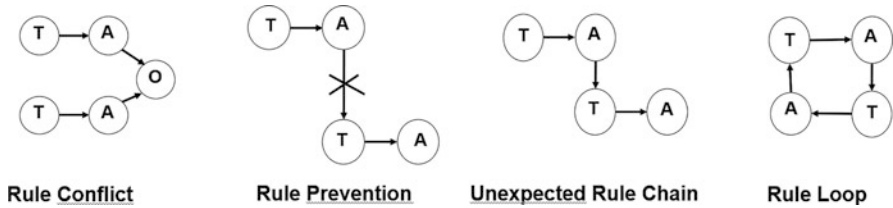


Fig. 10 Possible issues with multiple automation rules

Various conflicts can occur with such rules. For example, after 21.00 R1 and R2 provide conflicting indications regarding the windows. Rule prevention occurs when the execution of one rule unintentionally prevents the trigger of another rule, for example, if rule 1 is “if nobody is home, turn off the smart outlets” with the intention of saving energy, and rule 2 is “WHEN it’s 10AM, turn on the smart pet feeder.” If the pet feeder is powered from one of the smart outlets and nobody is home at 10 a.m., the pet will not be fed. An example of unexpected rule chain is if rule 1 is “IF the temperature in my room is below 20C, turn on the heater in my room,” and rule 2 is “IF the temperature in my room is above 25C, open the window.” After executing the first rule, the temperature might rise above 25C and activate rule 2, which can cause the window to open unexpectedly.

FORTNIoT (Coppers et al. 2020) is a tool that combines self-sustaining predictions (e.g., weather forecasts or sun position) and simulations of trigger-condition-action rules to deduce when these rules will trigger in the future and what state changes they will cause to connected smart home entities (e.g., light state changes). The predictions are represented through interactive timelines in which users can see contextual elements and the corresponding timeline appears indicating when they will change state, and the associated rules, indicating when they will be triggered because of some change in the contextual entities.

One possible way to support end-user debugging is presented in (Manca et al. 2019). It shows a solution in which on one side of the user interface the user can specify the relevant elements of the context of use, and on the other side a set of rules, and the tool indicates whether such rules can be executed in the considered context. Thus, rule execution is simulated in the indicated context of use (what/if) and it provides why/why not explanations for interactive debugging.

A contribution providing a more systematic set of questions that are relevant in explaining the behavior of intelligent systems has been presented by (Vera Liao et al. 2020). For this purpose, they indicate that it is important to consider various aspects:

- What kind of data does the system learn from?
- What kind of data does the system give?
- How accurate/precise/reliable are the predictions?
- How does the system make the prediction?
- Why/how is this instance given this prediction?

- Why/how is this instance not predicted?
- What would the system predict if this instance changes to?
- How should this instance change to get a different prediction?
- What is the scope of change permitted to get the same prediction?

Future research in EUD should indicate to what extent such questions are relevant and useful and how to support them in the composition process. Another associated question is to explain potential privacy and security-related issues associated with the automation rules. For example a rule indicating that any new photo taken by the smartphone camera should be posted on Facebook in some cases can raise privacy issues. For this purpose, Breve et al. (2023) have explored a solution exploiting keywords from the rule's component and the identified risks to construct a pretrained language model, which generates an automatic, component-dependent explanation when privacy or security risks are detected in a rule.

Augmented Reality-Based Support for Automation Control

Visual editors with conceptual representations of possible automation elements tend to be large, comprehensive, static, abstract EUD tools detached from the user's real context. It may not be straightforward to understand how to navigate the large number of elements that they represent. Understanding the elements and to what real element they refer to sometimes requires technical knowledge. Thus, there is a need for more narrowed, situated, dynamic representations associated with the physical objects available in the current user's context (Fig. 11).

Introducing augmented reality support for this purpose can help to overcome such issues. For this purpose, some useful requirements should be considered. It would be useful to avoid using special devices that many do not have, thus solutions based on smartphones have the potential to be easily adopted. It should provide users with the possibility of direct interaction with the object of interest, and monitoring nearby automations while moving, also at different levels of granularity, involving single objects or areas such as a room. Thus, users could have the ability to frame a real object directly and know the automations that involve it, create new automations, and modify existing automations.

A first solution in this direction is presented in (Ariano et al. 2023). It is based on an Android app, connected to the TAREME Platform, which uses Estimote beacons for the indoor localization, and object recognition through Vuforia Engine. It thus provides the opportunity to show information on the state of sensors and objects present in the current room as well as the associated automations, and create new automations or modifying existing ones. A different solution is presented in (Mattioli and Paternò 2023), which removes the need for a detection phase before interacting with the environment, and allows for more expressive rules. It still provides visualizations that support user control and understanding

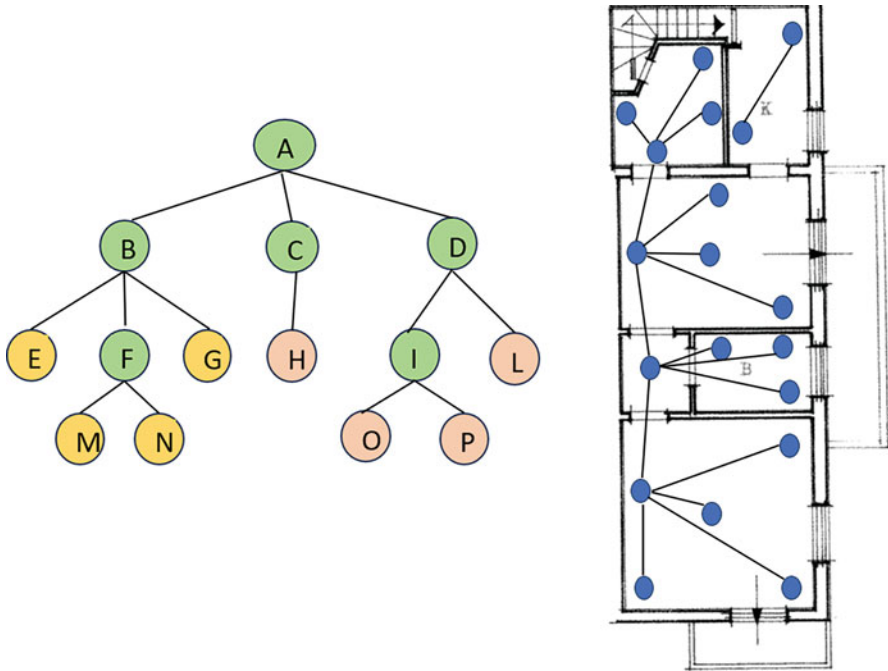


Fig. 11 From abstract representations to situated interaction

of the environment’s automations, also adds the possibility of providing rule recommendations that match the possible user intents in the rule creation phase, and allows the selection of external services to be accessed in relation to home objects.

Conclusion

In this chapter, we have seen how the explosion of the Internet of Things and AI has posed new challenges for the EUD field. The main one is how to allow users to create and control automations in the varied dynamic contexts in which they daily live. One further emerging technological trend is that associated with robots, which are becoming more and more present in many places characterizing daily life. In this process, they are appearing more social and similar to humans in their behaviors. As robots interact with a broader range of people, end-user robot interaction development is an important way to democratize robot adoption by empowering end users, who may not have experience in robot programming, to customize their behavior to meet users’ individual contextual needs. Robot programming is characterized by the embodied and situated nature of the technology being programmed, introducing

unique challenges such as the need for programs to reference and interact with the surrounding environment, which may potentially include task objects, obstacles, and people. Such robots can attract people's attention, engage them, and improve the user experience (UX) by expressing emotions, communicating through high-level dialogues, using natural cues, and exhibiting social skills and distinctive personalities. These capacities can allow the robots to be employed to interact more naturally and socially rather than be considered mere instrumental tools. Exploiting trigger-action rules for describing robot behaviors is possible (Leonardi et al. 2021), but it is not enough. Previous work (Brich et al. 2017) has highlighted that the trigger-action paradigm, even enhanced by AND/OR compositions, turns out to be good for modeling and programming basic dynamic situations, but is not well-suited for managing more complex scenarios and task automations (such as with humanoid robots), which need further, more expressive paradigms to support the different, ever-changing needs of users, who can have varying expertise (and interests) in technology and development.

Overall, our applications, objects, and robots can generate a myriad of possible events, and each user may have specific requirements about how to react to them. Such data can help us to obtain mixed-initiative, usable platforms for personalization of daily environments in order to improve user life experience. In such contexts, automations should be transparent to end users so that they can actually control and benefit from them. Design criteria for methods and tools to develop humanations are fundamental in this perspective. In particular, the area of explainable debugging, also in order to highlight potential security and privacy issues, conversational interfaces, and augmented reality-based support need to be better investigated in the near future. Last, but not least, the role of LLMs in facilitating development activities is important and is promising to radically change the way to approach them.

References

- Aghaee S, Pautasso C (2014) End-user development of mashups with natural mash. *J Vis Lang Comput* 25(4):414–432
- App Inventor MIT (2010). <http://info.appinventor.mit.edu/>
- Ariano R, Manca M, Paternò F, Santoro C (2023) Smartphone-based augmented reality for end-user creation of home automations, behaviour & information technology. Taylor and Francis, London
- Barricelli B, Cassano F, Fogli D, Piccinno A (2019) End-user development, end-user programming and end-user software engineering: a systematic mapping study. *J Syst Softw* 149:101–137, ISSN 0164-1212
- Brackenbury W, Deora A, Ritchey J, Vallee J, He W, Wang G, Littman M, Ur B (2019) How users interpret bugs in trigger-action programming. In: *Proceedings of the 2019 CHI conference on human factors in computing systems*, pp 1–12
- Breve B, Cimino G, Deufemia V (2023) Identifying security and privacy violation rules in trigger-action IoT platforms with NLP models. *IEEE IoT J* 10(6)
- Brich J, Walch M, Rietzler M, Weber M, Schaub F (2017) Exploring end user programming needs in home automation. *ACM Trans Comput Hum Interact* 24(2):11:1–11:35

- Carmien SP, Fischer G (2008) Design, adoption, and assessment of a socio-technical environment supporting independence for persons with cognitive disabilities. *Proc CHI*:597–606
- Chen X, Zhang X, Elliot M, Wang X, Wang F (2022) Fix the leaking tap: a survey of Trigger-Action Programming (TAP) security issues, detection techniques and solutions. *Comput Secur* 120, C (Sep 2022)
- Coppers S, Vanacken D, Luyten K (2020) FORTNIoT: intelligible predictions to improve user understanding of smart home behavior. In: *Proceedings of the ACM on interactive, mobile, wearable and ubiquitous technologies*, vol 4(4), Article No.: 124, pp 1–24
- Corcella L, Manca M, Paternò F, Santoro C (2017) Personalizing a student home behaviour. In: *Proceedings IS-EUD 2017*, LNCS 10303. Springer, pp 1–16
- Corno F, De Russis L, Roffarello AM (2021) Devices, information, and people: abstracting the internet of things for end-user personalization. In: *End-user development*. Springer International Publishing, Cham, pp 71–86. https://doi.org/10.1007/978-3-030-79840-6_5
- Cuccurullo S, Francese R, Risi M, Tortora G (2011) MicroApps development on mobile phones. In: Costabile M, Dittrich Y, Fischer G, Piccinno A (eds) *End-user development 6654*. Springer, Berlin/Heidelberg, pp 289–294
- Cypher A (ed) (1993) *Watch what i do: programming by demonstration*. MIT Press
- Cypher A, Dontcheva M, Lau T, Nichols J (2010) *No Code Required Giving Users Tools to Transform the Web*, Morgan Kaufmann, ISBN 978-0-12-381541-5
- Danado J, Paternò F (2012) Puzzle: a visual-based environment for end user development in touch-based mobile phones. In: *Human-centered software engineering. HCSE 2012, Lecture Notes in Computer Science*, vol 7623. Springer, Berlin/Heidelberg. https://doi.org/10.1007/978-3-642-34347-6_12
- Fischer G, Giaccardi E, Ye Y, Sutcliffe A, Mehandjiev N (2004) Meta-design: a manifesto for end-user development. *Commun ACM* 47(9):33–37. <http://doi.acm.org/10.1145/1015864.1015884>
- Gallo S, Paternò F (2022) A conversational agent for creating flexible daily automation. In: *Proceedings of the 2022 international conference on Advanced Visual Interfaces (AVI 2022)*. Association for Computing Machinery, Article 38, pp 1–8. <https://doi.org/10.1145/3531073.3531090>
- Gallo S, Manca M, Mattioli A, Paternò F, Santoro C (2021) Comparative analysis of composition paradigms for personalization rules in IoT settings. In: *Proceedings of the 8th international symposium on end user development, IS-EUD 2021, Lecture notes in computer science*, vol 12724. Springer, Cham. https://doi.org/10.1007/978-3-030-79840-6_4
- Gallo S, Malizia A, Paternò F (2023) Towards a Chatbot for Creating Trigger-Action Rules based on ChatGPT and Rasa. In: *Proceedings 4th international workshop on ‘Empowering People in Dealing with Internet of Things Ecosystems’ (EMPATHY 2023)*, CEUR workshop proceedings, vol 3408
- Ghiani G, Paternò F, Spano D (2009) Cicero designer: an environment for end-user development of multi- device museum guides. In: *Proceedings EUD’09*, Siegen, LNCS 5435, Springer, pp 265–274
- Ghiani G, Paternò F, Spano LD, Pintori G (2016) An environment for end-user development of web mashups. *Int J Human-Comput Stud* 87:38–64, Elsevier
- Huang J, Cakmak M (2015) Supporting mental model accuracy in trigger-action programming. In: *Proceedings of the 2015 ACM international joint conference on pervasive and ubiquitous computing (UbiComp’15)*. ACM, New York, pp 215–225. <https://doi.org/10.1145/2750858.2805830>
- Klokmoose C, Eagan J, Baader S, Mackay W, Beaudouin-Lafon M (2015) UIST’15: Proceedings of the 28th annual ACM symposium on user interface software & technology, pp 280–290. <https://doi.org/10.1145/2807442.2807446>
- Ko AJ, Abraham R, Beckwith L, Blackwell A, Burnett M, Erwig M, Scaffidi C, Lawrance J, Lieberman H, Myers B, Rosson MB, Rothermel G, Shaw M, Wiedenbeck S (2011) The state of the art in end-user software engineering. *ACM Comput Surv* 43(3):1–44. <https://doi.org/10.1145/1922649.1922658>

- Leonardi N, Manca M, Paternò F, Santoro C (2021) Trigger-action programming for personalising humanoid robot behaviour. In: ACM conference on human factors in computing systems (CHI'19), Glasgow, Paper 445. 1
- Leshed G, Haber EM, Matthews T, Lau TA (2008) CoScripter: automating & sharing how-to knowledge in the enterprise. CHI:1719–1728
- Liao V, Gruen D, Miller S (2020) Questioning the AI: informing design practices for explainable AI user experiences, CHI 2020
- Lieberman H, Paternò F, Klann M, Wulf V (2006) End-user development: an emerging paradigm. In: Lieberman H, Paternò F, Wulf V (eds) End user development. Springer, Dordrecht, pp 1–8. https://doi.org/10.1007/1-4020-5386-X_1
- Macías JA, Paternò F (2008) Customization of web applications through an intelligent environment exploiting logical interface descriptions. *Interacting with Computers Elsevier*, vol 20 (1), pp 29–47
- Manca M, Paternò F, Santoro C, Corcella L (2019) Supporting end-user debugging of trigger-action rules for IoT applications. *Int J Human-Comput Stud* 123:56–69
- Manca M, Paternò F, Santoro C (2021) Remote monitoring of end-user created automations in field trials. *J Ambient Intell Humaniz Comput*. <https://doi.org/10.1007/s12652-021-03239-0>
- Manca M, Paternò F, Santoro C (2022) End-user development in industrial contexts: the paper mill case study, *Behaviour & Information Technology*. Taylor & Francis. <https://doi.org/10.1080/0144929X.2022.2089597>
- Markopoulos P, Nichols J, Paternò F, Pipek V (2017) End-user development for the Internet of Things. In: ACM Transactions on Computer-Human Interaction (TOCHI), vol 24(2), 9, May 2017
- Matera M, Picozzi M, Pini M, Tonazzo M (2013) PEUDOM: a mashup platform for the end user development of common information spaces. *ICWE*:494–497
- Mattioli A, Paternò F (2021) Recommendations for creating trigger-action rules in a block-based environment. *Behav Inform Technol* 40(10):1024–1034
- Mattioli A, Paternò F (2023) A mobile augmented reality app for creating, controlling, recommending automations in smart homes. *Proceedings of the ACM on Human-Computer Interaction* 7(195):1–22
- Myers B, Buxton W (1986) Creating highly interactive and graphical user interfaces by demonstration. In: *Proceedings of SIGGRAPH '86*, vol 20, No. 4. ACM, Dallas, pp 249–258
- Nardi BA (1993) *A small matter of programming: perspectives on end user computing*. MIT Press, Cambridge, MA
- Nichols J, Lau T (2008) Mobilization by demonstration: using traces to re-author existing web sites. In: *Proceedings of the symposium on intelligent user interfaces*, pp 149–158
- Paternò F (2013) End user development: survey of an emerging field for empowering people. *ISRN Software Engineering*, vol 2013, Article ID 532659, 11 p. <https://doi.org/10.1155/2013/532659>
- Paternò F, Santoro C (2019) End-user development for personalizing applications, things, and robots. *Int J Human-Comput Stud* 131:120–130
- Resnick M, Maloney J, Monroy-Hernandez A, Rusk N, Eastmond E, Brennan K, Millner A, Rosenbaum E, Silver J, Silverman B, Kafai Y (2009) *Scratch: programming for all*. *Commun ACM* 52:60–67
- Salovaara A, Bellucci A, Vianello A, Jacucci G (2021) Programmable smart home toolkits should better address households' social needs. In CHI 21, May 8–13, 2021, Yokohama, Japan. ACM, New York, 14 p
- Srinivasan V, Koehler C, Jin H (2018) Rule selector: selecting conditional action rules from user behavior patterns. In: *Proceedings of the ACM Interaction Mobility Wearable Ubiquitous Technology*, 2, 1, Article 35 (March 2018), 34 p
- Ur B, McManus E, Pak Yong M, Littman M (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, pp 803–812. <https://doi.org/10.1145/2556288.2557420>

- Ur B, Yong Ho MP, Brawner S, Lee J, Mennicken S, Picard N, Schulze D, Littman ML (2016) Trigger-action programming in the wild: an analysis of 200,000 IFTTT recipes. In: Proceedings of the 34rd annual ACM conference on human factors in computing systems (CHI '16). ACM, New York, pp 3227–3231. <https://doi.org/10.1145/2858036.2858556>
- van Berkel N, Skov M, Kjeldskov J (2021) Human-AI interaction: intermittent, continuous, and proactive, *Interactions* 28 (6), 67–71
- Yang R, Newman M (2013) Learning from a learning thermostat: lessons for intelligent systems for the home. In: 2013 ACM international joint conference on Pervasive and ubiquitous computing, pp 93–102