Doctoral Dissertation
Doctoral Program in Computer and Control Engineering (32.th cycle)

End-User Development in the Internet of Things

Alberto Monge Roffarello
* * * * * *

Supervisor
Prof. Fulvio Corno

Politecnico di Torino
May 14, 2020

I hereby declare that the contents and organisation of this dissertation constitute my own original work and does not compromise in any way the rights of third parties, including those relating to the security of personal data.

. . . . . .

Alberto Monge Roffarello
Turin, May 14, 2020

Summary

In the contemporary Internet of Things (IoT) era, people can interact with a multitude of smart devices, always connected to the Internet, in the majority of today's environments. With lamps, thermostats, and many other appliances that can be remotely controlled, homes and workplaces are becoming "smart." Furthermore, by using PCs and smartphones, users can access a variety of online services, ranging from social networks to news and messaging apps. The result is a complex network of *connected entities*, be they physical devices or virtual services, that can communicate with each other, with humans, and with the environment. This complex scenario opens up, at the same time, possibilities and issues.

By taking advantage of End-User Development (EUD) solutions, users can actively participate in the IoT by *personalizing* the functionality of their connected entities. Nowadays, in particular, many different visual programming platforms such as IFTTT and Zapier allow the personalization of the joint behaviors of connected entities through IF-THEN rules, i.e., in the form of *"if something happens on a device or a service, then execute an action on another device or service."* The growing spread of new smart devices and online services, however, makes this personaliziation a complex task, especially for users without programming experience. The trigger-action programming paradigm, indeed, is typically implemented at a low-level of abstraction, with representation models that strongly depend on the exploited technologies. This negatively influences the rule definition process: end users experience difficulties in finding and managing the functionality they are interested in, and they are likely to introduce dangerous run-time errors in the defined IF-THEN rules.

Stemming from these issues, this thesis presents a set of research works that aim at assisting end users in easily and efficiently personalizing the functionality of their connected entities. Through rigorous user studies and controlled experiments, we report on different approaches and practical solutions to *a*) simplify the definition of IF-THEN rules, *b*) promote the discovery of new rules and related functionality, and, *c*) enable the identification and the resolution of run-time problems in IF-THEN rules. For supporting these results, we first define a semantic representation, named *EUPont* (**E**nd-**U**ser **P**rogramming **ont**ology), to model abstract and

technology-independent IF-THEN rules that can be adapted to different contextual situations. We demonstrate that *EUPont* is more expressive than the representation models offered by contemporary trigger-action programming platforms, and that it improves the processes needed by end users to define IF-THEN rules in terms of time, understandability, and ease of use.

After presenting *EUPont*, we then report on how we used it to facilitate users in discovering and managing rules and related functionality. We present, in particular, two different approaches, namely *EUDoptimizer* and *RecRules*. *EUDoptimizer* is an optimization tool that adopts semantic optimization methods to dynamically redesign layouts in trigger-action programming user interfaces on the basis of the choices made by the user during the definition of a rule. The tool is based on *SDP-FSM*, a predictive model for trigger-action programming that exploits *EUPont* and a state-of-the-art model of human performance in menu search named Search-Decision-Pointing. *RecRules*, instead, is an innovative recommender system of IF-THEN rules that, by exploiting *EUPont*, is able to compute suggestions on the basis of the final behaviors users would like to define, e.g., increasing the temperature in a room.

Finally, we explore the urgent need of assessing the correctness of IF-THEN rules by presenting two end-user debugging tools, i.e., *EUDebug* and *My IoT Puzzle*. By exploiting *SCPN*, i.e., a novel formalism based on Petri Nets and the *EUPont* model, such tools are able to assist users in identifying possible loops, inconsistencies, and redundancies that their rules may generate at run-time. *EUDebug*, in particular, is built on top of an IFTTT-like interface, while the *My IoT Puzzle* exploits the Jigsaw metaphor, and it has been designed on a set of guidelines extracted from previous work on end-user debugging in different contexts.

Summarizing, the main outcomes of this thesis are the following:

EUPont, an ontological high-level representation for end-user development that allows the definition of abstract and technology-independent IF-THEN rules that can be adapted to different contextual situations, independently of manufacturers, brands, and other technical details.

EUDoptimizer, an optimization tool to dynamically redesign layouts in trigger-action programming interfaces in an interactive way, i.e., by considering the choices made by end users during the definition of a rule.

RecRules, a hybrid and semantic recommendation system of IF-THEN rules that allows users to discover new rules on the basis of the underlying functionality, rather than the involved brands or manufacturers.

EUDebug, an end-user debugging tool built on top of an IFTTT-like interface that enables end users to debug their IF-THEN rules at definition time.

My IoT Puzzle, an end-user debugging tool to compose and debug IF-THEN rules though the Jigsaw metaphor, designed according to a set of guidelines extracted from the literature.

Acknowledgements

I would like to thank you my supervisor, Fulvio Corno, and Luigi De Russis for their continuous support and mentorship. Their guidance helped me in becoming a better researcher.

A special thanks to all the current and past members of the e-Lite research group, and, in particular, to Juan Pablo Saénz, with whom I have shared this PhD journey.

Thank you to my parents, Elisa e Paolo, and my girlfriend, Cecilia, for their patient and unconditional support over these years.

Contents

List of Tables x

List of Figures xi

1 Introduction 1
- 1.1 Defining IF-THEN Rules . . . 1
  - 1.1.1 Rule Definition Process . . . 2
  - 1.1.2 Identified Issues . . . 3
- 1.2 Thesis Contributions . . . 4
  - 1.2.1 Thesis Organization . . . 7

2 End-User Development in the IoT: an Overview 9

3 Moving Towards a Higher Level of Abstraction 13
- 3.1 Simplifying Trigger-Action Programming: a High-Level Semantic Approach . . . 14
  - 3.1.1 Background . . . 15
  - 3.1.2 *EUPont* Desing and Implementation . . . 17
  - 3.1.3 Model Expressiveness . . . 24
  - 3.1.4 User Evaluation . . . 26
  - 3.1.5 Results . . . 33
- 3.2 Discussion and Guidance for Future Research . . . 36

4 Discovering IF-THEN Rules and Functionality 39
- 4.1 *EUDoptimizer*: Defining IF-THEN Rules with an Optimizer in the Loop . . . 40
  - 4.1.1 Background . . . 41
  - 4.1.2 Optimizing IF-THEN Rule Definition . . . 41
  - 4.1.3 SDP-FSM: A Predictive Model for Trigger-Action Programming . . . 43
  - 4.1.4 Optimization Problem and Methods . . . 47
  - 4.1.5 Implementation and Technical Assessment . . . 48
  - 4.1.6 User Evaluation . . . 54

4.1.7 Results . . . 56
4.2 Recommending IF-THEN Rules for End-User Development . . . 58
4.2.1 Background . . . 59
4.2.2 Recommending By Functionality . . . 62
4.2.3 Knowledge Graph Model & Problem Formulation . . . 63
4.2.4 The *RecRules* Algorithm . . . 65
4.2.5 Algorithm Evaluation . . . 72
4.2.6 Results . . . 77
4.3 Discussion and Guidance for Future Research . . . 82

5 End-User Debugging in Trigger-Action Programming 87
5.1 Run-Time Problems in IF-THEN Rules . . . 88
5.1.1 Background . . . 88
5.1.2 Characterizing Problems . . . 89
5.1.3 Modeling and Detecting Problems . . . 91
5.1.4 SCPN RESTful Server . . . 96
5.2 Exploring End-User Debugging in Trigger-Action Programming Platforms . . . 96
5.2.1 Background . . . 98
5.2.2 The *EUDebug* Tool . . . 98
5.2.3 User Evaluation . . . 99
5.2.4 Results . . . 104
5.3 Debugging IF-THEN Rules Through the Jigsaw Metaphor . . . 109
5.3.1 Background & Adopted Technologies . . . 110
5.3.2 Extracting Design Guidelines . . . 110
5.3.3 The *My IoT Puzzle* System . . . 113
5.3.4 User Evaluation . . . 115
5.3.5 Results . . . 118
5.4 Discussion and Guidance for Future Research . . . 119

6 Conclusions 123
6.1 Summary of Contributions . . . 123
6.2 Future Works . . . 125

A Publications 127
A.1 International Journals . . . 127
A.2 Proceedings . . . 128
A.3 Book Chapters . . . 129

Bibliography 131

List of Tables

1.1 Thesis contributions and main outcomes 5
3.1 The main OWL classes modeled in the *EUPont* ontology 20
3.2 Ur et al. dataset [137] description 25
3.3 Translation of IFTTT rules in the *EUPont* representation 26
3.4 General demographics in the *EUPont* user study 29
3.5 Quantitative results of the *EUPont* user study 33
4.1 Results of Simulated Annealing and Ant Colony System for trigger layouts 51
4.2 Results of Simulated Annealing and Ant Colony System for action layouts 53
4.3 Statistics of the dataset exploited in the *RecRules* evaluation 73
4.4 Example of a rule used in the *RecRules* evaluation 73
4.5 Distribution of the computed Graded Implicit Feedback (GIF) in the *RecRules* evaluation 74
4.6 Comparative results of the three learning to rank algorithms in implemented in *RecRules* 78
4.7 Diversity, coverage, and serendipity results in the *RecRules* evaluation 78
4.8 Accuracy comparison of *RecRules* with other state-of-the-arts algorithms 80
4.9 Diversity, coverage, and serendipity comparison of *RecRules* with other state-of-the-arts algorithms 82
5.1 A list of IF-THEN rules analyzed by *EUDebug* 93
5.2 General demographics in the *EUDebug* user study 101
5.3 The 12 trigger-action rules defined in the *EUDebug* study 102
5.4 Quantitative results in the *EUDebug* user evaluation 104
5.5 Qualitative results in the *EUDebug* user evaluation 108
5.6 Design guidelines for end-user debugging tools for trigger-action programming 111
5.7 Quantitative results in the *My IoT Puzzle* user evaluation 118

List of Figures

1.1 The complex network of connected entities with which users interact daily . . . 2
1.2 The definition process to define an IF-THEN rule . . . 3
3.1 Connected Entity Selection in IFTTT and Zapier . . . 13
3.2 The *EUPont* structure . . . 18
3.3 How the individual instances of the *EUPont*'s OWL classes can be linked together . . . 19
3.4 Temperature-related actions in *EUPont* . . . 21
3.5 Location-related triggers in *EUPont* . . . 22
3.6 Rule modeling in *EUPont* . . . 23
3.7 The *IFTTT-like* interface implemented for testing *EUPont* at definition time . . . 27
3.8 The *EUPont* interface implemented for testing *EUPont* at definition time . . . 28
4.1 An overview of *EUDoptimizer* . . . 42
4.2 Some screenshots of the user interface used in the empirical evaluation. The interface resembles IFTTT and allows the definition of trigger-action rules with both the IFTTT version and the *EUDoptimizer* enhanced version. In the IFTTT terminology, rules are named applets, while connected entities are named services. . . . 50
4.3 Trigger layout for defining triggers: IFTTT *vs. EUDoptmizer* . . . 52
4.4 Optimized action layout calculated by *EUDoptimizer* . . . 54
4.5 General time performances in the*EUDoptimizer* user study . . . 57
4.6 Time performances for uncommon rules in the*EUDoptimizer* user study . . . 57
4.7 The knowledge graph model exploited by *RecRules* . . . 64
4.8 *RecRules* architecture . . . 65
4.9 An example of a knowledge graph built by *RecRules* . . . 66
4.10 The Semantic Linking & Graph Instantiation phase of *RecRules* . . . 67
4.11 The Semantic Reasoning Process phase of *RecRules* . . . 69
4.12 Collaborative path in *RecRules* . . . 70
4.13 Technology path in *RecRules* . . . 70

4.14 Functionality path in *RecRules* 71
5.1 The elements characterizing the SCPN formalism 92
5.2 A SCPN example 94
5.3 The SCPN RESTful server architecture 96
5.4 An overview of the *EUDebug* tool 97
5.5 The user interface of *EUDebug* for defining a new IF-THEN rule . . 99
5.6 *EUDebug* Problem Checking 100
5.7 *EUDebug* Step-by-Step Explanation 100
5.8 Quantitative results for direct and indirect in the *EUDebug* user evaluation 108
5.9 Two mockups produced in designing *My IoT Puzzle* 114
5.10 Defining an IF-THEN rule with *My IoT Puzzle* 115
5.11 Real time feedback provided by *My IoT Puzzle* 116
5.12 Resolving a problem with *My IoT Puzzle* 117

Chapter 1

Introduction

The Internet of Things (IoT) is the paradigm whereby everyday objects are no longer disconnected from the virtual world, but they can be controlled remotely and can serve as an access point to the Internet [95]. The advent of the IoT already helps society in many different ways, through applications ranging in scope from the individual to the planet [25]. People, in particular, can nowadays interact with a multitude of smart devices: with lamps, thermostats, and many other appliances, including fridges and ovens, that can be connected to the Internet, homes are becoming "smart." Other environments as well, ranging from workplaces to entire cities, are extensively leveraging on the IoT. A smart environment, in particular, can be defined as a "small world where all kinds of smart devices are continuously working to make inhabitants' lives more comfortable [29]." Besides physical devices, many different online services, ranging from social networks to news and messaging apps, are greatly used by almost everyone: the number of people using the Internet has passed 4.5 billion marks in January, 2020, with more than 3.8 billion people actively using social media [128]. As a result, users can easily access a complex network of *connected entities* (Figure 1.1), be they smart devices or online services, that can communicate with each other, with humans, and with the environment.

1.1 Defining IF-THEN Rules

In the scenario offered by such a complex network, the End-User Development (EUD) vision [87] aims at putting customization mechanisms in the hands of end users, i.e., the subjects who are most familiar with the actual needs to be met. Starting from iCAP [47], an early rule-based system for building context-aware applications, several works in the literature demonstrated the effective applicability of EUD techniques for the personalization of the functionality of smart devices and online services in different areas, including mobile environments [105], smart

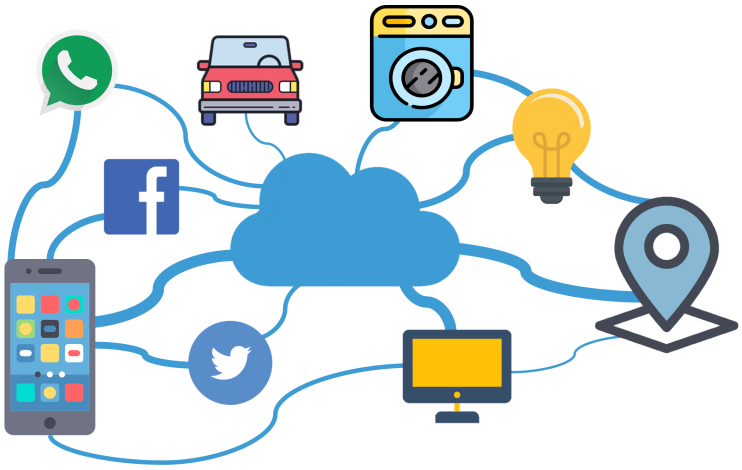


Figure 1.1: The complex network of connected entities with which users interact daily. Besides interacting with smart devices in different contexts, e.g., lamps and cars, users can easily access a variety of online services such as Facebook and WhatsApp through their smartphones.

homes [136, 16], and web mashups [130, 43]. Nowadays, end users can take advantage of visual programming platforms such as IFTTT [72] and Zapier [151] to personalize the *joint* behaviors of their own connected entities, without the need of writing any code. Most of these platforms [46], in particular, adopt the trigger-action programming paradigm, i.e., they allow the definition of IF-THEN rules such as

- if I publish a photo on *Facebook*, then upload it to my *Google Drive*;
- if the *Nest* security camera detects a movement, then blink the kitchen's *Philips Hue* lamp;
- if the *Nest* thermostat detects that the temperature rises above 22 Celsius degrees, then open the *SmartThings* window.

1.1.1 Rule Definition Process

In contemporary trigger-action programming platforms, users can define IF-THEN rules through similar wizard-based procedures [46]. Figure 1.2 summarizes the *rule definition process* to be followed. Defining a rule means defining, separately, a trigger and an action to be linked together.

To define a trigger, the following steps must be completed:

1. select, through some menus, the generic connected entity involved in the rule's trigger (*Connected Entity Selection*). In this step, entities are typically modeled through the underlying manufacturer or brand, e.g., *Nest* thermostat.

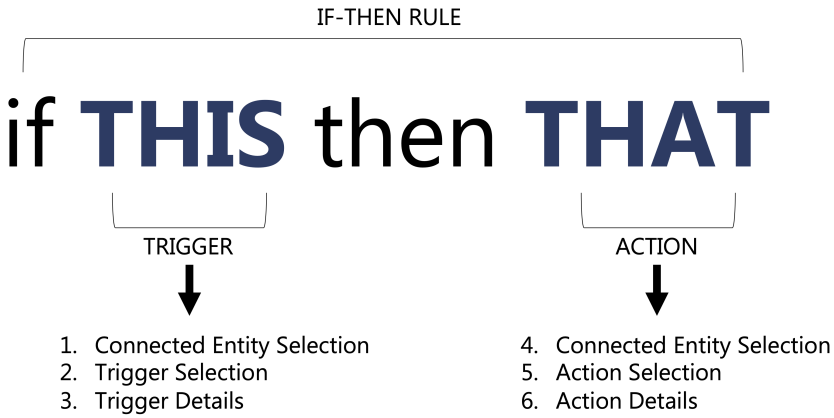


Figure 1.2: The definition process that users must follow to define an IF-THEN rule in the most common trigger-action programming platforms, e.g., IFTTT and Zapier.

2. select, through some menus, the specific trigger to be monitored (*Trigger Selection*), e.g., the detection of a high temperature;

3. complete the trigger with additional details (*Trigger Details*), e.g., the identifier of the specific *Nest* thermostat, or a temperature threshold.

The same three steps need to be repeated to define the rule's action (steps 4,5, and 6 in Figure 1.2).

1.1.2 Identified Issues

Ideally, the trigger-action programming paradigm and can express most of the behaviors desired by potential users [136]. Unfortunately, despite its wide adoption, the way it is implemented nowadays presents its own set of problems. We identify, in particular, 3 main issues related to the definition of IF-THEN rules with contemporary trigger-action programming platforms, namely *low-level of abstraction*, *information overload*, and *run-time problems*.

Low-Level Abstraction. In the forthcoming IoT world, new "things" will not always be knowable a priori [152] but they may appear and disappear at every moment, also depending on user location (e.g., as with public services in a smart city). Unfortunately, contemporary trigger-action programming platforms adopt highly technology-dependent representation models that poorly adapt to the increasing complexity of the IoT ecosystem: they work with well-know connected entities, previously associated to a specific user, only. Two smart devices or online services that provide equivalent or identical functions

(e.g., setting the indoor temperature) but differ in brands or manufacturers, in particular, are currently treated like distinct entities. Therefore, as the number of available connected entities grows and varies, the complexity of the IoT ecosystem grows [9], and defining IF-THEN rules becomes a complex task for non-programmers [71].

Information Overload. With the low-level of abstraction of contemporary trigger-action programming platforms, i.e., every device and online service modeled on the basis of the underlying brand or manufacturer, the number of possible combinations among triggers and actions of different technologies is high, and the number of rules shared on these platforms is growing. Zapier, for example, supports more than 1,000 devices and web applications, each one with its own triggers and actions, while the number of publicly available and reusable rules on IFTTT already exceeded 200,000 in September, 2016 [137]. Unfortunately, contemporary trigger-action programming platforms do not provide users with any discovery support [137], and the explosion of new smart devices and online service results in user interfaces with too much information.

Run-Time Problems. Another important and urgent challenge is the need to provide users with instruments for understanding and debugging their IF-THEN rules, i.e., to avoid possible conflicts [21] and to assess the rules' correctness [46]. Indeed, given the low-level of abstraction of contemporary trigger-action programming platforms, users frequently misinterpret the behavior of trigger-action rules [17], often deviating from their actual semantics, and are prone to introduce errors [70]. Errors in this context, however, can lead to unpredictable and dangerous behaviors [17]: while posting a content on a social network twice could be considered a trivial issue, the wrong rules could unexpectedly unlock the main door of a house, thus generating a security threat. Unfortunately, even if providing end users with validation features and warning mechanisms could facilitate the adoption of EUD solutions in the real world [46], relatively little work has been done in this area.

1.2 Thesis Contributions

Stemming from the aforementioned issues, this thesis presents a set of research works at the intersection of EUD, trigger-action programming, and IoT to assist end users in easily and efficiently personalizing the functionality of their connected entities. Table 1.1 summarizes the contributions of this work and its main outcomes. We organized them according to 3 different categories, one for each identified issue, i.e., *low-level abstraction*, *information overload*, and *run-time problems*, respectively.

Table 1.1: Thesis contributions and main outcomes for each identified issue, i.e., low-level abstraction, information overload, and run-time problems.

Category	Thesis Contributions	Main Outcomes
Higher Abstraction	• An ontological high-level representation for end-user development in the IoT. • The implementation of a user interface to define abstract and technology independent IF-THEN rules.	• *EUPont*
Discovery	• A predictive model for trigger-action programming to optimize user interfaces of trigger-action programming platforms. • A tool to dynamically redesign layouts in trigger-action programming user interfaces. • A hybrid and semantic recommendation algorithm of IF-THEN rules.	• *SDP-FSM* • *EUDoptimizer* • *RecRules*
Debugging	• The formal characterization of the control-flow problems that may arise in IF-THEN rules at run-time. • A novel formalism based on Petri Nets and semantic information to model and check the run-time behavior of IF-THEN rules. • A set of guidelines extracted from the literature to design user interfaces for end-user debugging in the trigger-action programming context. • Two different end-user debugging tools for trigger-action programming.	• *SCPN* • *EUDebug* • *My IoT Puzzle*

Moving Towards a *Higher* Level of *Abstraction*

To overcome the *low-level abstraction* issue, we envisioned a new breed of programming environments able to support a "higher level" representation of smart devices and online services, with the aim of simplifying the rule definition process. To this end, we designed and implemented *EUPont*, an ontological high-level representation for end-user development that allows the definition of abstract and technology-independent IF-THEN rules that can be adapted to different contextual situations, independently of manufacturers, brands, and other technical details. We integrated the model in a trigger-action programming user interface, by testing it in a controlled user study with 30 participants. Results demonstrate that *EUPont* is more expressive than the representation models offered by contemporary trigger-action programming platforms, and it introduces several benefits in the rule definition process. By defining IF-THEN rules such as *"if I enter a closed space, then cool the environment"*, users are not requested to specify technological details, and they can personalize the functionality of their connected entities with fewer rules, fewer mistakes, and in less time.

Discovering IF-THEN Rules and Functionality

As a response to the *information overload* issue, we supported the need of providing users of trigger-action programming platforms with more support for discovering and managing new rules and related functionality [137]. To this end, we explored two different approaches. First, we designed and implemented an optimization tool, named *EUDoptimizer*, that dynamically reorders the layouts in trigger-action programming user interfaces in an interactive way, i.e., by considering the choices made by users during the rule composition phase. The aim is to promote the discovery of the "right" connected entity to be used for defining the trigger or the action, according to the current user need. For the optimization problem, we defined *SDP-FSM*, a predictive model for trigger-action programming based on *EUPont* and a state-of-the-art model of human performance in menu search. Results of a user study with 12 participants suggest that *EUDoptimizer* can help end users define IF-THEN in less time, by reducing the cognitive effort needed in the rule definition process.

In the second approach, we built on the idea that information overload in trigger-action programming platforms could be addressed through recommender systems. To this end, we proposed *RecRules*, a hybrid and semantic recommendation system of IF-THEN rules. By exploiting the *EUPont* model, it allows users to discover new rules on the basis of the underlying functionality, rather than the involved brands or manufacturers. A rule for turning on a *Philips Hue* lamp, for example, is *functionally* similar to a rule for opening the *Hunter Douglas* blinds, because they

share a common final goal, i.e., to light up a place. Results from different experiments demonstrate the benefits of using semantic information in the recommendation process, and show that *RecRules* outperforms state-of-the-art recommendation algorithms.

End-User *Debugging* in Trigger-Action Programming

To mitigate the *run-time problems* issue, we claim that users should be assisted in identifying programming bugs in IF-THEN rules and reason about how to fix them during the rule definition process, as already highlighted by previous work [14]. This implies designing trigger-action programming platforms that provide users with mechanisms to debug their IF-THEN rules. We started by formally characterizing the control-flow problems that may arise in IF-THEN rules at run-time, i.e., loops, inconsistencies, and redundancies. Then, we defined *SCPN*, a novel formalism based on Petri Nets and *EUPont* to model the run-time behavior of IF-THEN rules and identify possible problems.

We used the SCPN formalism in two different end-user debugging tools for trigger-action programming, namely *EUDdebug* and *My IoT Puzzle*. We designed and implemented *EUDdebug* on top of an IFTTT-like interface: it enables end users to debug their IF-THEN rules at definition time by *a*) assisting them in identifying rule conflicts, and *b*) allowing them to foresee the run-time behavior of their rules through step-by-step simulation. Results from a controlled user study with 15 participants show evidence that users can successfully face computer-related concepts such as loops, inconsistencies, and redundancies with the help of *EUDebug*. The step-by-step simulation, in particular, helps users understand why their rules might generate a specific problem.

To improve the user interface of *EUDdebug*, we then reviewed previous work on end-user debugging in different contexts, e.g., spreadsheets and web mashups, and we extracted a set of design guidelines. Based on this analysis, we designed and implemented *My IoT Puzzle*, an end-user debugging tool to define and debug IF-THEN rules that exploits the Jigsaw metaphor. The tool interactively assists users in the definition process by representing triggers and actions as complementary puzzle pieces, and by providing real-time feedback to test *on-the-fly* the correctness of the rule under definition. Results of a preliminary user study with 6 participants suggest that the usage of different representations and visual languages facilitates users in analyzing problems and helps them understand, identify, and correct errors in IF-THEN rules.

1.2.1 Thesis Organization

The thesis is organized as follow:

Chapter 2 presents related works on EUD and trigger-action programming in the IoT.

Chapter 3 focuses on the *low-level abstraction* issue, and it speculates on the usage of a higher level of abstraction to simplify the end-user definition of IF-THEN rules. It presents, in particular, *EUPont* and its evaluations (Section 3.1).

Chapter 4 focuses on the *information overload* issue, by exploringhow to assist users in discovering rules and related functionality. It presents *EUDoptimizer* (Section 4.1), including the exploited *SDP-FSM* predictive model (Section 4.1.3), and *RecRules* (Section 4.2).

Chapter 5 focuses on the *run-time problems* issue, and it investigates the need of empowering users in debugging IF-THEN rules. It presents the *SCPN* formalism (Section 5.1), the two different end-user debugging tools, i.e., *EUDebug* (Section 5.2) and *MyIoTPuzzle* (Section 5.3), and the guidelines to design trigger-action programming user interfaces (Section 5.3.2).

Chapter 6 concludes the thesis and presents future works.

Chapter 2

End-User Development in the IoT: an Overview

The IoT already changed the way end users use the Internet, as well as mobile and sensor-based devices: people are increasingly moving from passive consumers to active producers of information, data, and software [103]. They can access new building blocks and tools, analogously to what happened with blogs and wikis during the early phases of the Web [38]. As demonstrated by previous works [46, 60], the IoT facilitates the creation of heterogeneous ecosystems wherewith users can access functionality and data offered by the so called "smart objects [4]," i.e., interconnected physical devices equipped with electronics, sensors, and actuators. In this context, end users are willing to link together the different "behaviors [46]" exposed by such devices, with the aim of accommodating their everyday needs. According to Ghiani et al. [60], in particular, end users can be considered as the most suitable stakeholder to specify how the available smart devices should be exploited to create new valuable applications: they know what is required to build applications that can support their tasks, and the availability of new technologies increases their motivation to participate in the creation of applications that satisfy their needs [52]. Furthermore, the wide adoption of online services such as social networks and messaging apps has further expanded the possibility of creating applications in various domains [137].

Providing users with efficient End-User Development (EUD) methodologies and tools to customize the behavior of their connected entities, be they physical smart objects or online services, is therefore an urgent challenge. According to Lieberman et al. [87], EUD can be 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 artifact*".

Integration of IoT technologies with online services and applications through end-user programming environments allows users to effectively participate in the IoT [42]. The research community, especially in the HCI and ubiquitous computing

fields, started to explore the possibilities offered by EUD more than 10 years ago. One of the first works in this domain is iCAP [47], a visual, PC-based, and rule-based system for building context-aware applications that does not require users to write any code. Nowadays, EUD approaches and methodologies have been already extensively explored in different contexts, e.g., mobile environments [105], smart homes [136, 16], and web mashups [130, 43].

One of the most popular paradigm to empower end users in directly programming their connected entities is the trigger-action programming [136, 47]. Trigger-action programming offers a very simple and easy to learn solution for creating end-user applications, according to Barricelli and Valtolina [9]: it is not surprising that, in the last years, several commercial trigger-action programming platforms were born with the aim of allowing end-user personalization of connected entities. Examples include IFTTT [72], Zapier [151], Microsoft Flow [100], Mozzilla's Thing Gateway [141], SmartRules [127], and many others. In its basic form, trigger-action programming allows users to connect a single event to a single action: by defining trigger-action (IF-THEN) rules, users can connect a pair of devices or online services in such a way that, when an event (the *trigger*) is detected on one of them, an *action* is automatically executed on the latter. Although some behaviors would require greater expressiveness to be defined in a single rule, e.g., through multiple actions or additional trigger conditions, many of the most popular trigger-action programming platforms, e.g., IFTTT, Zapier, and Microsoft Flow, still continue to adopt the basic form of the trigger-action programming paradigm [14].

In this thesis, we focus on IF-THEN rules with a single trigger and a single action. Despite the presented models and tools can be easily generalized to include more expressive versions of the trigger-action programming paradigm, our aim was to avoid unnecessary difficulties for the users involved in our studies. Indeed, while some studies found that users are able to define rules with multiple triggers, conditions, and actions [136], others demonstrated that users often misinterpret the behavior of rules with more complex triggers and actions [70], e.g., because they do not understand the differences between states, instantaneous triggers, and conditions. Moreover, despite apparent simplicity, even the process of composing IF-THEN rules with single events and single actions has been found to be a complex task for non programmers [71], and the expressiveness and understandability of platforms like IFTTT have been criticized since they are rather limited [136, 70, 137]. Barricelli and Valtolina [9] analyzed the most popular end-user tools for personalizing connected entities, including IFTTT, and found that some of them "offers a too complex solution for supporting end-users in expressing their preferences." To better assist users in defining personalization in the evolving IoT scenario, the authors presented an extension of the trigger-action paradigm that incorporated not only devices, sensors, and online services, but also recommendation systems, other IoT users, space and time, and the social dimension. Ur

et al. [136] found that the trigger-action approach can be both useful and usable for end-user development in IoT settings like smart homes, but they also found that the level of abstraction end users employ to express triggers needs to be better explored. Their paper investigated the practicality of end-user programming for customizing smart home devices, by evaluating thousands of trigger-action rules publicly shared on IFTTT, and conducting a usability test with more than 200 participants. They found that many users express triggers one level of abstraction higher, e.g., "when I am in the room" instead of "when motion is detected by the motion sensor." In another study, Ur et al. [137] empirically analyzed more than 200,000 IFTTT public rules, the largest-scale investigation of this type up to now, finding that a large number of users are crafting a diverse set of IF-THEN rules, which represents a very broad array of connections for filling gaps in devices and services functionality. According to the authors, this explosion of entities and connections highlights the need to provide users with more support for discovering functionality and managing collections of IF-THEN rules. The analysis emphasizes also the future need of making "IFTTT rules more expressive." Similarly, Huang and Cakmak [70] systematically studied the impact of different trigger and action types in trigger-action programming environments, focusing their efforts on IFTTT. Two user studies revealed inconsistencies in interpreting the behavior of trigger-action programming and some errors in creating programs with a desired behavior. This highlights the need of assisting users in assessing the correctness of their rules, e.g., through debugging approaches [46]. As IF-THEN rules are deployed in increasingly complex scenarios, indeed, users must be able to identify programming bugs and reason about how to fix them, as highlighted by Brackenbury et al. [14]. In their analysis on how users interpret bugs in trigger-action program, the authors provided a specification of the trigger-action programming model, by identifying ten programming bugs that might arise in IF-THEN rules. They found 10 types of problems, by classifying them into control-flow bugs (e.g., infinite loops), timing bugs (e.g., non-deterministic timing), and inaccurate user expectations (e.g., priority conflicts). They also showed that eight out of the ten identified bugs negatively influence users' ability to correctly predict the outcomes of IF-THEN rules. While the work of Brackenbury et al. is an important contribution to understand trigger-action programming bugs, however, the question on how to help users with these problems is still underexplored.

To solve the aforementioned issues, several recent works explored new approaches to empower end users in programming their connected entities. Huang and Cakmak offered four recommendations for improving the IFTTT interface with the aim of mitigating the issues that arise from mental model inaccuracies: (a) to include prompts for warning users in ambiguous situations; (b) to disallow confusing options; (c) to better distinguish event and state triggers when they are related to the same underlying concept (e.g., "it starts raining" and "it is raining"); and

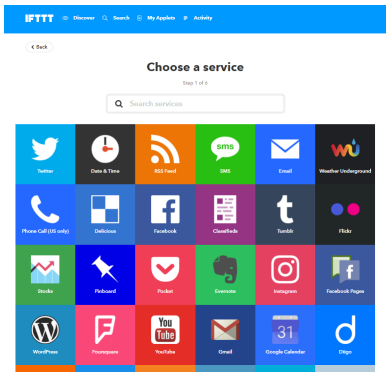
(d) to consider higher level program statements alternatives to "if" and "then." Danado and Paternò developed Puzzle [42], a mobile framework which allows end users without IT background to create, modify, and execute applications. Brich et al. [16] reported on the comparison of two different notations, i.e., rule-based and process-oriented, in the smart home context, showing that trigger-action rules are generally sufficient to express simple automation tasks, while processes fit well with more complex tasks. Akiki et al. [1] presented ViSiT, an approach that allows end users to specify transformations on IoT objects that are automatically converted into executable workflows. Desolda et al. [46] reported on the results of a study to identify possible visual paradigms to define trigger-action rules in the IoT. They proposed a model that includes new operators for defining rules, by combining multiple events and conditions exposed by smart objects. The authors also presented the architecture of a platform to support rules execution. The architecture was composed of three layers, i.e., interaction layer, logic layer, and service layer. The separation of concepts enables the definition of multiple front-ends addressing different execution platforms, i.e., different devices. Ghiani et al. [60] proposed a method and a set of tools for end users to personalize the contextual behavior of their IoT applications through trigger-action rules. The authors, in particular, described a generic model and its specialization in a home automation use case, evaluating it during the rule definition process.

Our work starts from the issues and the opportunities presented in this Chapter. We explored, in particular, how to improve the definition of IF-THEN rules by taking advantage of abstract representations of triggers and actions, discovery mechanisms, and debugging features, with the aim of better coping with the evolving IoT scenario.

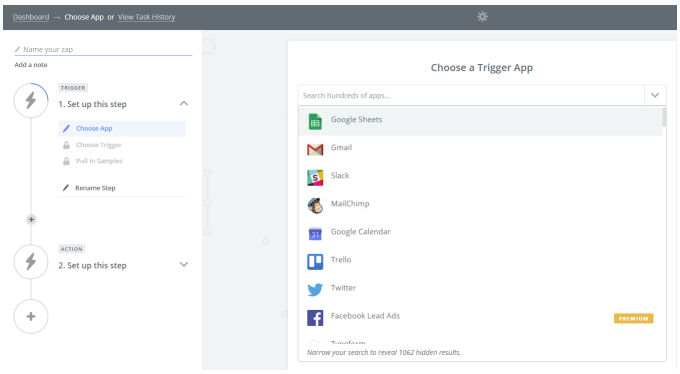
Chapter 3

Moving Towards a Higher Level of Abstraction

Contemporary trigger-action programming platforms adopt highly technology-dependent representation models that poorly adapt to the increasing complexity of the IoT ecosystem (*low-level abstraction* issue). In the case of IFTTT and Zapier, for example, devices and services in the Connected Entity Selection step of the rule definition process (Section 1.1.1) are simply grouped by manufacturer or brand, as shown in Figure 3.1. As a result, the definition of IF-THEN rules becomes a complex task for non-programmers [71].



(a) Connected entity selection in IFTTT



(b) Connected entity selection in Zapier

Figure 3.1: Connected Entity Selection in IFTTT (a) and Zapier (b). The first step in the rule definition process is the selection of the type of device or online service involved in the rule's trigger. Such a selection is typically performed by searching in large menus of supported products. With the spread of new supported technologies, the amount of information may become too high, thus making the rule definition process difficult.

Take John, for example:

> *John, a manager of an insurance company, is always hot, especially in summer. He loves air conditioning, and he would like to set a low temperature wherever it is possible. At home, John has an intelligent Nest thermostat that he controls through his Android smartphone. John goes to work by car. There, all the offices are equipped with a Samsung smart air conditioner.*

By exploiting the procedures and the representation models provided by contemporary trigger-action programming platforms, *John* has to define several IF-THEN rules to reach his comfort goal, at least one for his home, one for his office, and one for his car, even if they perform the same logical operations (i.e., set a specific temperature when he enters a place). Furthermore, he has to be aware of every single technology he may encounter before creating his rules (e.g., *Nest*, *Samsung*, etc.), to choose the right one for each rule. Finally, even with an authorization, *John* will not be able to define similar rules for unknown places or "things" (e.g., his friend's car), i.e., similar rules do not adapt to different contexts.

With such a "low-level" of abstraction, the user experience with contemporary trigger-action programming platforms is put to a hard test. Remembering and maintaining all the specific rules that users are forced to define is challenging, especially when users' needs change over time. To simplify the definition of IF-THEN rules, we envisioned a new breed of programming environments that are designed to support a "higher level" representation of smart devices and online services. Our idea was to allow *John* to define a single rule for his need, e.g., *"if I enter a closed space, then cool the environment"*.

To this end, we designed and implemented *EUPont* (**E**nd-**U**ser **P**rogramming **ont**ology), a high-level representation for End-User Development. Part of the work described in this chapter has been previously published in several papers. A generic overview of the approach can be found in [30] and [102]. The detailed description of the *EUPont* ontology (Section 3.1), along with its user evaluation (Section 3.1.4), are based on the work published in [31]. The evaluation of the *EUPont* expressiveness (Section 3.1.3) is based on the work published in [32].

3.1 Simplifying Trigger-Action Programming: a High-Level Semantic Approach

To take a step towards a higher level of abstraction in trigger-action programming platforms, *EUPont* allows users to model abstract and technology-independent IF-THEN rules. Such rules can be adapted to different contextual situations, independently of manufacturers, brands, and other technical details. The representation

abstracts the IoT ecosystem by modeling connected entities on the basis of their functionality. Through semantic and reasoning capabilities, in particular, *EUPont* is able to link real devices and services to the abstract behaviors they can execute, thus providing a strong support for the run-time execution of the high-level rules.

3.1.1 Background

EUPont is a semantic representation designed as an OWL ontology. OWL ontologies are a fundamental part of the Semantic Web framework used for formally defining classes, attributes, and relationships between the concepts in a specific domain. The OWL language[1], in particular, is characterized by a formal semantic, and it is built upon the World Wide Web Consortium's (W3C) XML standard for objects called the Resource Description Framework (RDF)[2]. Generally speaking, an OWL ontology is composed of a set of *classes*, i.e., the concepts describing the modeled domain, that can be instantiated through *individuals*. Individuals can be linked together by means of *object properties*, while *data properties* can be used to associate particular attributes, e.g., a name or a numerical value, to each individual.

We chose to exploit the Semantic Web framework to design *EUPont* for four main reasons.

Reasoning capabilities: semantic reasoning can be used to infer information that has not been explicitly told about, thus facilitating the mapping between abstract information to the low-level details needed to actually execute high-level rules.

Data integration and reuse: the continuous growing of the IoT ecosystem raises the question of how new connected entities can be easily integrated in existing trigger-action programming platforms. Semantic technologies offer *by nature* advantages in terms of data reuse and integration, thus making it easy to integrate new devices and online services.

Meaningful information: in a semantic model, and, in particular, in a OWL ontology, data is enriched with semantic information, i.e., meaning. Thanks to such a semantic, we can easily perform queries on the representation such as *"which smart devices or online services can perform a particular action?"* or *"which connected entity can generate a particular event?"*

Concept hierarchies: a semantic model is described as a graph that embeds inheritance relationships among concepts, thus allowing the definition and the linking of multiple levels of abstraction with ease.

[1] https://www.w3.org/TR/owl2-overview/, last visited on November 18, 2019

[2] https://www.w3.org/RDF/, last visited on November 18, 2019

Adding semantics to IoT systems is a topic of particular interest in the literature: researchers agree that the IoT could benefit from a semantic approach in terms of interoperability, data integration, and knowledge extraction [8], and the lack of open and shared IoT standards naturally leads to a semantic-oriented perspective [4]. Indeed, contemporary IoT systems have an app-centric or device-centric approach towards their users, as they are too often designed with an industry-centered approach that promotes vertical silos [103]. IoT adopters can control their own smart thermostat with a dedicated app, they are able to query information from their cars with another app, and they can turn their connected lamps on and off with yet another app. As a result, all of these web or smartphone-based apps are often neither intuitive nor efficiently usable [96].

Semantics in the IoT has mainly been adopted in a very specific area, i.e., for describing sensors and their capabilities. One of the most significant and widespread models in use in this field is the Semantic Sensor Network (SSN) [27], an ontology to describe sensors, observations, and features of interest. Other contributions in this area have been proposed by the Open Geospatial Consortium (OGC), that developed a set of XML-based standards to describe sensors and their related data [13]. The expressiveness of these vocabularies and ontologies allows them to be used on a very wide range of applications. However, as reported by Bermudez et al. [10], they are too specific, and the description of non-essential components for many use cases can make the ontologies heavy to query and process. To tackle this issue, they proposed IoT-lite [10], a lightweight instantiation of SSN that allows interoperability and discovery of sensory data in heterogeneous IoT platforms.

The IoT, however, is not composed by sensors, only. Unfortunately, few previous works included in their models other concepts, such as users, on-line services, interfaces, etc. In the IOT-A project[3], the authors identified entities, resources, and services as key concepts of the IoT domain. In [45], the authors proposed a modeling approach in which IoT resources are able to expose standard service interfaces. Similarly, Wang et al. [140] exploited the concept of services to extend SSN and to represent other IoT-related concepts such as actuators, gateways, and servers.

Even with the introduction of such new concepts to sensor modeling, all the aforementioned vocabularies represent the IoT with a device and technology-oriented perspective. This approach does not entirely take into account contemporary IoT ecosystems, where categories (e.g., lighting systems, temperature systems, etc.) and final capabilities of connected entities (e.g., "can this lighting device be turned on?") are a fundamental information.

Such information is partially taken into account by some previous works in the field of smart environments. With DogOnt [12], the authors presented a building

[3] *http://www.iot-a.eu/public* (last visited on November 11, 2016)

modeling ontology designed to fit real world home automation system capabilities and to support interoperation between currently available and future solutions. By exploiting reasoning capabilities, DogOnt is able to describe, for example, where a device is located, the set of capabilities of such a device, and the technology-specific features needed to interface the device. In the same field, on behalf of the European Commission, the TNO[4] organization developed SAREF[5], a shared model of consensus that facilitates the matching of existing assets (standards, protocols, data-models, etc.) in the smart appliances domain. Mayer et al. [96], instead, presented a high-level description language that captures the semantic of the interactions provided by smart devices, with the aim of implementing intuitive interfaces for remote control. The idea is to enrich smart devices with a minimal amount of markup representing *a)* the type of information that they can exchange, and *b)* the high-level semantics of the provided interaction, e.g., set or get a value.

EUPont is inspired by the aforementioned smart environment models, but it is specifically designed to support abstract and technology-independent IF-THEN rules in the broader context of connected entities personalization.

3.1.2 *EUPont* Desing and Implementation

In the *EUPont* design process, we carefully reviewed the content and the structure of existing trigger-action programming platforms as well as the reported issues and challenges from the literature. In particular, we analyzed the IFTTT and Zapier platforms, i.e., the supported connected entities with their available triggers and actions, to find high-level behaviors and possible common functionality. Then, we grouped the triggers and the actions of the different devices and services according to each identified behavior. For example, we obtained the behavior "*set thermostat temperature*" by grouping different actions of 20 diverse devices (e.g., Caleo, ecobee, Nest, tado Smart AC Control, Wink Aros, etc.). Stemming from such an analysis, we finally developed the *EUPont* ontology by using Protégé[6], a free and open-source OWL ontology editor for building intelligent systems. Moreover, we adopted HermiT[7] as the semantic reasoner. The resulting ontology is available at `http://elite.polito.it/ontologies/eupont.owl`.

[4] `https://www.tno.nl/en/` (last visited on November 27, 2019)

[5] `http://ontology.tno.nl/saref` (last visited on November 27, 2019)

[6] `https://protege.stanford.edu` (last visited on November 27, 2019)

[7] `http://www.hermit-reasoner.com/` (last visited on November 27, 2019)

EUPont Architecture

Figure 3.2 shows the general structure of *EUPont*. The ontology is composed of three layers, which are interlinked to support both the definition and the execution of abstract and technology-independent IF-THEN rules:

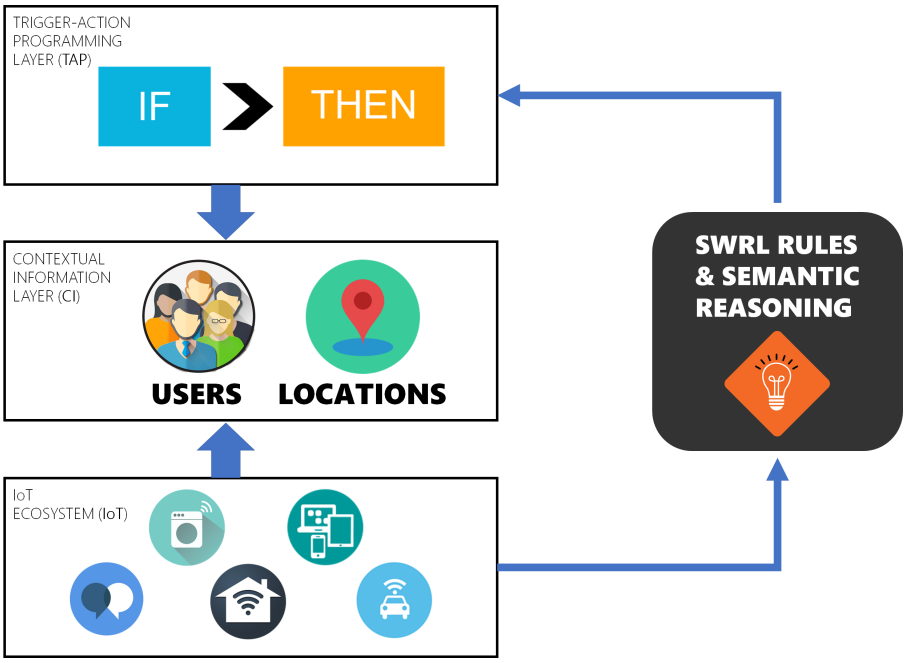


Figure 3.2: The *EUPont* structure. The Trigger-Action Programming and the IoT Ecosystem layers refer to the same Contextual Information, and are linked together thanks to a set of SWRL rules.

Trigger-Action Programming (TAP) Layer allows the definition of abstract IF-THEN rules, that are independent from manufacturers, brands, or any other technological details. It defines a hierarchy of triggers and actions to be used for defining rules in the trigger-action form.

IoT Ecosystem (IoT) Layer models smart devices and online services on the basis of their categories (e.g., environment systems, user devices, etc.) and final capabilities (e.g., switching capabilities, communication capabilities, etc.).

Contextual Information (CI) Layer describes locations and users that are shared between the TAP and the IoT layers, e.g., the position of a device, or the users subscribed to an online service.

The three layers are linked together in two ways:

1. The IoT layer and the TAP layer are both connected to the same CI layer. For example, a device is linked with its current position, and an action of a rule can be linked with the location in which the action has to be performed.

2. The TAP layer can be directly connected to the IoT layer. A **semantic reasoning** process, in particular, automatically maps the defined *EUPont* rules with real entities in the IoT layer *able* to reproduce the desired abstract behaviors, according to the capabilities of the available devices and services.

EUPont Axiomatisation

Each ontology layer is composed of a series of top-level OWL classes (reported in Table 3.1), while individual instances of such classes can be connected together by means of different object properties, as shown in Figure 3.3.

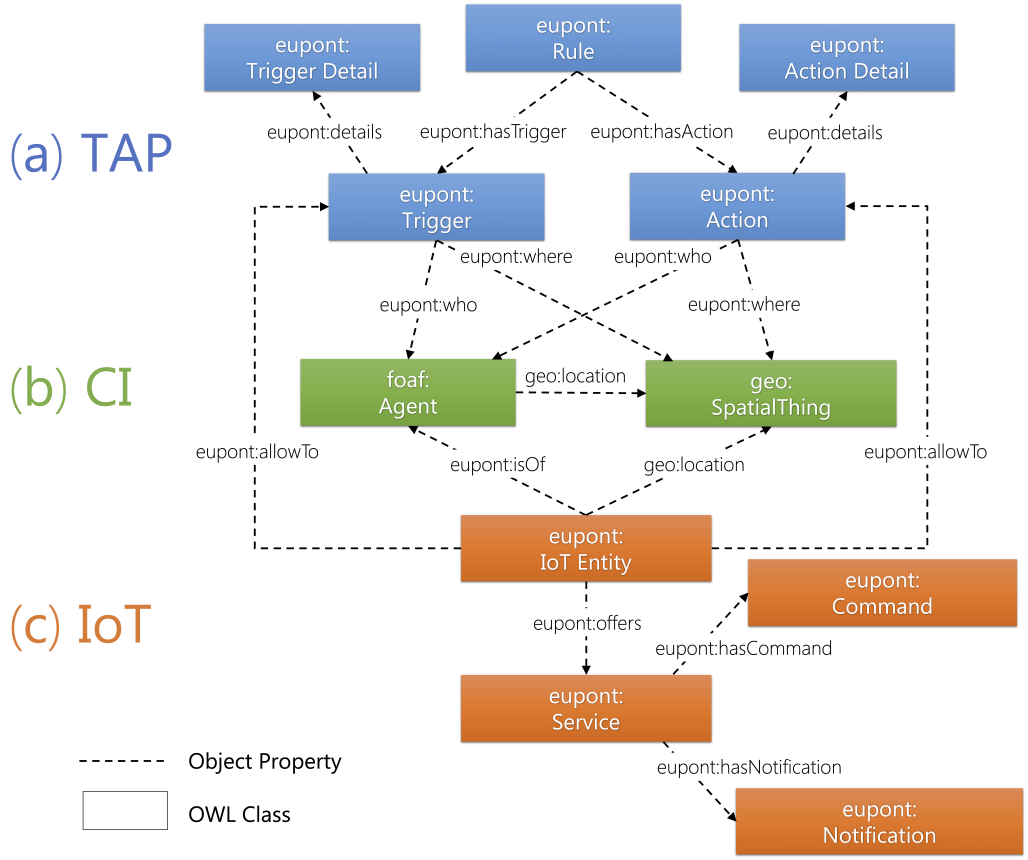


Figure 3.3: How the individual instances of the *EUPont*'s OWL classes can be linked together. The figure clearly shows the layered structure of the ontology.

The TAP layer is composed by OWL classes representing IF-THEN `Rule`(s),

Table 3.1: The main OWL classes modeled in the *EUPont* ontology.

Class	Description	Layer
`Rule`	A Rule in the trigger-action form.	TAP
`Trigger`	The trigger of an IF-THEN rule, i.e., an event to react to. It is specialized in a hierarchy of OWL subclasses representing events expressed at different level of abstraction.	TAP
`Action`	The action of an IF-THEN rule, i.e., an action to perform when the trigger of the rule occurs. It is specialized in a hierarchy of OWL subclasses representing actions expressed at different level of abstraction.	TAP
`Trigger Detail`	A detail to be associated with a trigger, e.g., a temperature threshold.	TAP
`Action Detail`	A detail to be associated with an action, e.g., a message to be sent.	TAP
`Agent`	A user in the modeled ecosystem. The class is imported from the FOAF ontology.	CI
`SpatialThing`	A location involved in the modeled ecosystem. The class is imported from the W3C Basic Geo Vocabulary.	CI
`IoT Entity`	A hierarchy of connected entities (smart devices or online services) classified by their categories (e.g., environment systems, user devices, social networks, etc).	IoT
`Service`	A service (i.e., a capability) exposed by a connected entity.	IoT
`Command`	A specific low-level action that can be executed by a connected entity. It includes the technology-specific features needed to interface the entitiy.	IoT
`Notification`	A specific low-level event that can be generated by a connected entity. It includes the technology-specific features needed to interface the entitiy.	IoT

`Trigger`(s), `Actions`(s), and related details (Figure 3.3, (a)). As suggested by previous work [46], a rule can have multiple triggers and multiple actions (`hasTrigger` and `hasTrigger` object properties, Figure 3.3). In this way, the model can be easily generalized to different versions of the trigger-action programming paradigm. To allow end users to choose their preferred level of abstraction, triggers and actions are organized hierarchically by functionality in two levels:

High-Level (HL) trigger and action classes model *generic* event to be verified or actions to be executed, respectively, and they do not include any technical detail, *nor* the type of device or service to be used to implement the desired behavior.

Medium-Level (ML) trigger and action classes model *specific* events to be verified or actions to be executed, respectively, and they *allow* the specification

of the generic devices or services type to be used, without including any technological detail.

As an example, Figure 3.4 shows a partial view of the hierarchical tree that characterizes temperature-related actions. `Increase` and `Lower` temperature are HL actions, since the behaviors they define can be achieved in different ways, e.g., by turning the heater on or off, or by closing or opening a window. If an end user is interested in better specifying the desired operation, she can use ML actions, which for temperature-related actions include `Turn Heater On`, `Close the Window`, etc.

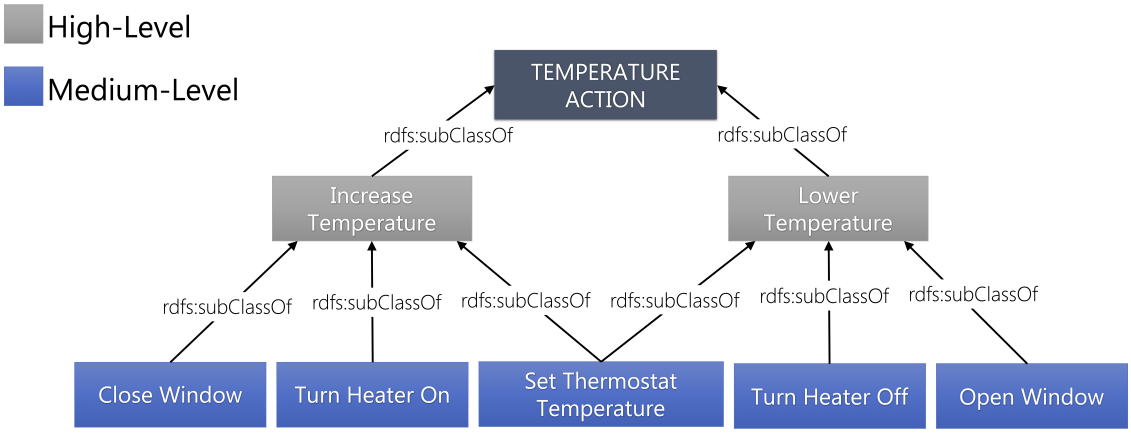


Figure 3.4: A partial view of the hierarchical class tree that characterizes temperature-related actions.

Figure 3.5, instead, shows some location-related triggers. `Enter Place` and `Exit Place` are HL triggers, since the event of entering or leaving a place can be monitored in different ways and through different connected entities. ML location-related triggers, for example, model a door that has been opened, or a camera that detects a presence.

As shown in Figure 3.3, each trigger and action can have different details (`details` object property), and it can be connected to contextual information (Figure 3.3, (b)) by means of the `who` and `where` object properties. We modeled users and locations by reusing established ontologies and vocabularies, as suggested by well-known ontology development guidelines [50]. Users can be instantiated as individuals of the `Agent` FOAF[8] class. For locations, instead, we specialized the `SpatialThing` class of the W3C Basic Geo Vocabulary[9]. We added subclasses needed to describe the locations involved in IoT ecosystems, e.g., buildings (`Bulding`) and rooms (`Room`), as well as cars (`Car`) and other vehicles.

[8] http://www.foaf-project.org/ (last visited on November 11, 2019)

[9] http://www.w3.org/2001/sw/interest/ (last visited on November 11, 2019)

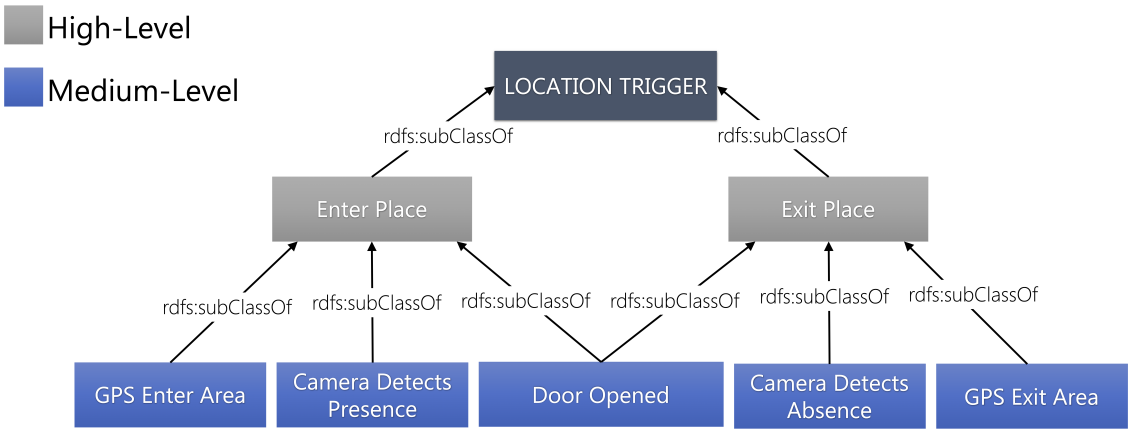


Figure 3.5: A partial view of the hierarchical class tree that characterizes location-related triggers.

Locations and users, i.e., the CI layer, can also be connected to the IoT layer (Figure 3.3, (c)) thtough the `isOf` and `location` object properties. The IoT layer, in particular, is composed of different OWL classes that describe connected entities. The `IoT Entity` class is specialized in various subclasses that represent different device and service types, e.g., `Thermostat`, `Lamp`, `Social Network`, etc. Each `IoT Entity` individual `offers` one or more `Service`, that represent a particular capability. Services may have `Command`(s), i.e., actions that can be executed, and `Notification`(s), i.e., events that can be registered (`hasCommand` and `hasNotification` object properties, respectively). `Command`(s) and `Notification`(s) classes, in particular, capture the semantics of the provided interaction, e.g., to set or get a particular environmental parameter, as in the work of Mayer et al. [96].

Finally, the `allowTo` object property of Figure 3.3 represents the result of the reasoning process that directly connects IF-THEN rules in the TAP layer with real devices and services able to reproduce the desired behaviors. The reasoning process is supported by a set of SWRL rules, which are in charge of dynamically instantiating `allowTo` properties between `IoT Entity` individuals and `Trigger` and `Action` individuals. Such an *ability* connection, along with the information stored on the shared CI layer, can be used at run-time by trigger-action programming platforms to execute *EUPont* rules onto real connected entities, according to the contextual situation. In case of multiple options, different solutions could be adopted to select the final connected entities in charge of executing the abstract *EUPont* rules. As proposed in the discussion of this work (Section 3.2), a possibility is to provide users with multi-level interfaces exposing the hierarchy of possible triggers and actions, from the highest level of abstraction to triggers and actions with more specific details. Other solutions range from preference-based approaches, where the user explicitly declares her preference towards specific devices and online services, to

fully automatic solutions, e.g., by using machine learning algorithms. A particular type of an automatic solution that uses the abstract triggers and actions of *EUPont* to extract IF-THEN rules in the IFTTT representation is reported in Section 4.2.

EUPont in Practice

To exemplify the proposed semantic approach, let us consider a user, i.e., John, and the following IF-THEN rule:

> `RULE_123` *"if I enter home, then set the thermostat temperature to 23 Celsius degree"*

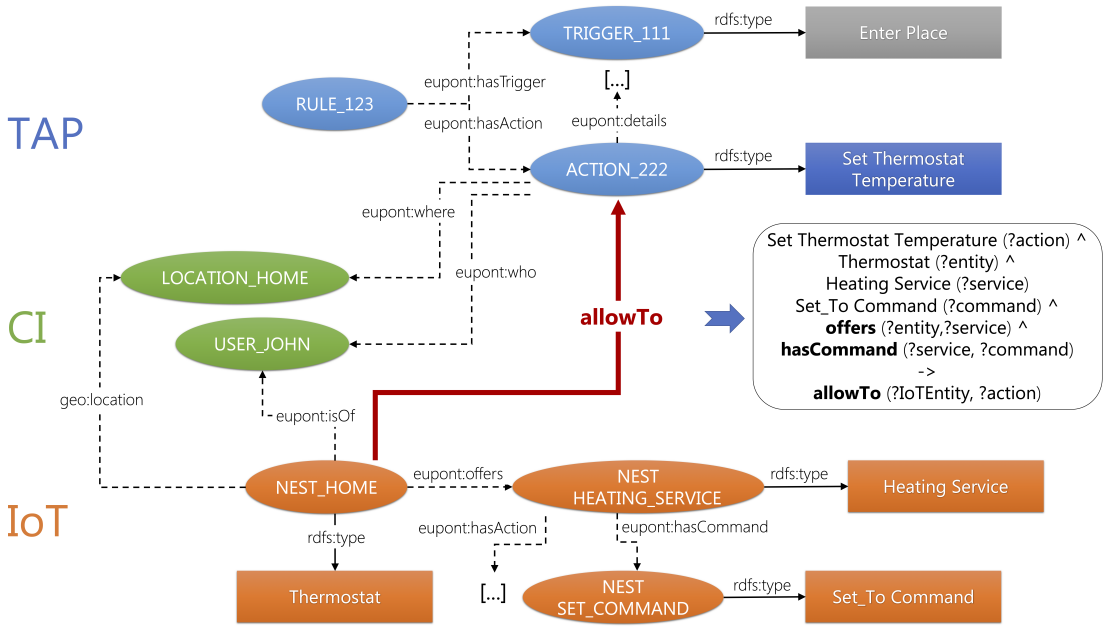


Figure 3.6: Graphical representation of the rule *"if I enter home, then set the thermostat temperature to 23 Celsius degree"* as modeled in *EUPont*. Thanks to the reported SWRL rule, the action is linked to a real device able to reproduce it.

The rule has exactly one trigger and one action. According to the *EUPont* model, the trigger is expressed with a High-Level of abstraction, since it does not include any technical detail, nor the type of device or service to be used to detect the event. The action, instead, is expressed with a Medium-Level of abstraction: it mentions, at least in a generic way, a thermostat. Figure 3.6 partially shows how `RULE_123` is modeled in *EUPont*, and how *EUPont* supports its real time execution.

a) The trigger is of type `Enter Place`, a HL class, while the action is an instance of the `Set Thermostat Temperature` ML class (`ACTION_222`). The action,

in particular, is connected to John, i.e., the rule's creator, and the John's home, respectively.

b) The John's home is equipped with a `NEST_HOME` smart thermostat that offers a `Heating Service`. Such a service allows to set a target temperature through a `Set_To Command`.

c) *EUPont* exploits the SWRL rule reported in Figure 3.6 to link `NEST_HOME` to `ACTION_222`. Following the SWRL rule, in particular, any `Thermostat` that offers a `Heating Service` with a `Set_To Command` is automatically considered able to reproduce `Set Thermostat Temperature` actions, i.e., an `allowTo` object property is automatically instantiated between `NEST_HOME` and `ACTION_222`. Since the `Set Thermostat Temperature` is a ML action, the same `NEST_HOME` thermostat is also automatically able to reproduce the HL behaviors that are parents of that action in the TAP hierarchy, e.g., `Increase Temperature`.

3.1.3 Model Expressiveness

As a first evaluation of our *EUPont* model, we focused on its expressiveness, i.e., the set of different triggers and actions it allows to represent. To this end, we compared it with the representational models currently adopted in trigger-action programming, by investigating the following research questions:

RQ1) Is *EUPont* at least as expressive as the representations used by contemporary trigger-action programming platforms? Does it allow to represent the same behaviors expressed by low-level triggers and actions?

RQ2) Is *EUPont* compatible with low-level trigger-action rules defined with contemporary trigger-action programming platforms?

To answer these questions, we exploited a dataset of IF-THEN rules publicly shared on IFTTT as of September 2016 [137], by trying to categorize all the involved triggers and actions in their corresponding ML and HL *EUPont* classes. Two researchers were involved in this manual mapping. The resulting ontology is available at `http://elite.polito.it/ontologies/eupont-ifttt.owl`. Table 3.2 (column "Original dataset") reports some statistics about the original dataset. For each of the 295,156 rules, the dataset includes:

- id, creation date, description, and number of shares of the rule;
- trigger name, description, and channel (i.e., involved device or service);
- action name, description, and channel (i.e., involved device or service).

Table 3.2: Ur et al. dataset [137] description.

	Original dataset	**After pre-processing**
Rules	295,156	290,963
Distinct Triggers	976	951
Distinct Actions	551	528
Users	129,206	127,173
Rules for each user (in average)	2.29 (SD=6.5)	2.29 (SD=6.5)
Max number of rules for a user	982	982
Min number of rules for a user	1	1

Before translating the IF-THEN rules in the *EUPont* representation, we performed a data pre-processing step. Since we were interested in the final behaviors of the defined rules, we identified in the dataset the rules composed of ambiguous triggers or actions in terms of functionality. For example, the rule *"if the Wemo switch is turned on, then send me an Android SMS"* has an ambiguous trigger, because we do not know which devices are connected to the switch. For all the identified ambiguous rules, we manually inspected the description field, trying to discover more information about actual devices involved in triggers and actions and the user's intent. We deleted from the dataset 4,193 rules for which it was impossible to resolve the ambiguity. Finally, the pre-processed dataset was composed of 290,963 rules composed by 127,173 different users (Table 3.2, column "After pre-processing").

After data pre-processing, we carried out the translation process in three distinct phases. First, we developed a Java program that uses the OWL API library [67] to automatically instantiate and connect IFTTT triggers, actions, and rules in the *EUPont* ontology. Then we manually mapped each instantiated trigger and action to one or more `Trigger`(s) or `Action`(s) ML classes, e.g., `Close Window` and `Turn Heater Off` (Figure 3.4). Finally, we redefined the IFTTT rules by replacing their trigger and action individuals with the corresponding ML and HL `Trigger`(s) and `Action`(s) versions, respectively, by removing all the duplicates resulting from the translation towards a higher level of abstraction.

As an example, the dataset rule

- if the entrance *Nest Cam* recognizes me, then turn on the kitchen *Philips Hue* lamp (**IFTTT**)

was translated as follow:

- if the kitchen's management system detects a presence, then turn the kitchen's lights on (**Medium-Level**);
- if I enter the kitchen, then illuminate it (**High-Level**);

Table 3.3: Translation of IFTTT rules in the *EUPont* representation.

	Medium-Level	**High-Level**
Number of translated rules	290,963	290,963
Number of rules after translation	179,110	170,353
Number of spared rules	111,853	120,610
% of spared rules	37.90%	41.45%
% of spared rules per user	12.26% (SD=24.12%)	13.26% (SD = 25.18%)

Table 3.3 reports the results of the translation process with both ML and HL classes. All the 290,963 IFTTT rules in the pre-processed dataset were translated. With respect to the original dataset, *EUPont* allowed to translate 98.58% of the rules, along with 97.44% of triggers and 95.83% of actions. Furthermore, as already explained, the 4,193 ambiguous rules were mainly due to a lack of information in the original dataset: by knowing the functionality defined by the users, also those rules could be translated.

The translation confirms that *EUPont* could significantly reduce the number of rules needed by end users to satisfy their needs. With the Medium-Level translation, the total number of rules is reduced by 37.90%, and, in average, each user could save the 12.26% of their rules. By expressing the rules in an even more abstract way, i.e., with the High-Level of abstraction, the percentage of saved rules increases (41.45% in total, and with an average saving of 13.26% for each user). Moreover, such promising results are influenced by the dataset distribution. In fact, 87,796 users (i.e., 68%) share one rule, only. In this case, obviously, the translation does not influence the final number of rules for the users but may avoid the creation of similar rules in the future. These findings show that *EUPont* is as least as expressive than the representation model used by IFTTT (**RQ1**), and it is fully compatible with low-level rules as defined in IFTTT (**RQ2**). Moreover, the flexible trigger-action approach adopted by *EUPont* (e.g., with multiple triggers and actions) increases the expressiveness of the representation.

3.1.4 User Evaluation

After investigating the *EUPont* expressiveness, we conducted a user study to evaluate the suitability and the understandability of the *EUPont* approach by end users. The user study was a controlled in-lab experiment that involved 30 participants, of which 15, only, had programming experience. It focused on the creation of IF-THEN rules both with the current representation of IFTTT and the *EUPont* representation, i.e., ML and HL triggers and actions. The study addressed the following research questions:

RQ3) Does the *EUPont* representation help users creating their IF-THEN rules

more effectively and efficiently compared with the IFTTT representation?

RQ4) Which of the two representations is preferred by users, and which are the perceived advantages and disadvantages of the two solutions?

To carry out the study, we built two versions of a graphical interface modeled after IFTTT. Whereas our *IFTTT-like* interface resembled the representation adopted by IFTTT (Figure 3.7), but with a limited number of supported entities, our *EUPont* interface allowed users to create IF-THEN rules with the *EUPont* representation, i.e., through ML and HL triggers and actions (Figure 3.8). For the *EUPont* interface, in particular, we used HL classes for triggers and ML classes for actions. We made this choice according to previous the work of Ur et al. [137], that shows that users are typically more abstract when referring to event, while they are more specific in defining actions.

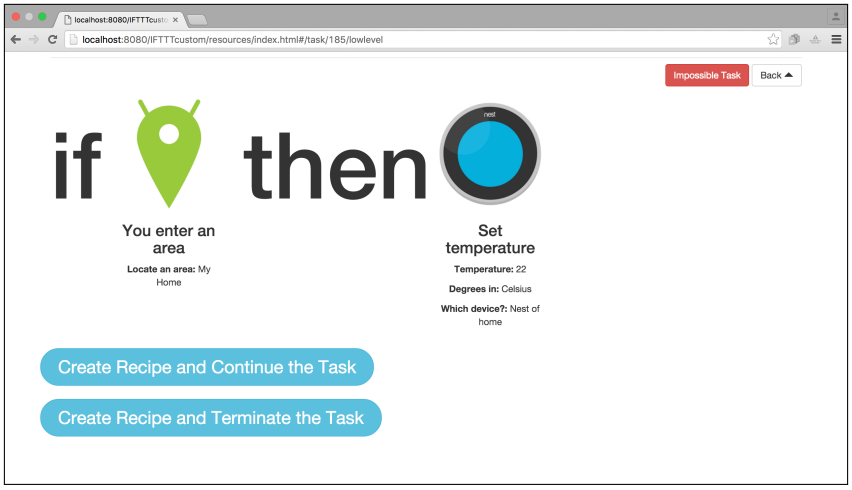


Figure 3.7: The study interface in the *IFTTT-like* representation, showing the recipe "If I my Android smartphone detects that I enter the home area, then set my Nest thermostat to 22 Celsius degree."

The study was composed of five tasks related to the definition of IF-THEN rules. All the participants performed all the five tasks twice, once with the *IFTTT-like* and once with the *EUPont* interface. We followed a two-way mixed design. We considered the used representation (*IFTTT-like* or *EUPont*) as the within-subject factor, and the participant group (users with or without programming experience) as the between-subject factor. The experiment was carried out in an office of our university, and took about 1 hour per participant. Two tasks out of five were carried out with the think-aloud protocol. In this case, we asked participants to vocalize their thought process as they performed the tasks. All study sessions were video recorded and observation notes were taken by the researchers. System activities and associated data were logged as well.

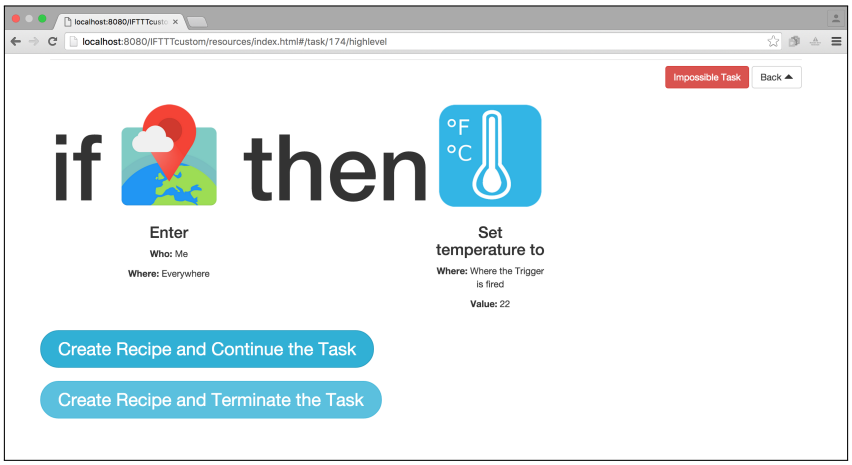


Figure 3.8: The study interface in the *EUPont* representation, showing the recipe "If I enter home, then set the temperature to 22 Celsius degree."

Participants

The study involved 30 participants. To avoid the introduction of biases in the population sample, we recruited two different groups of university students by considering their formal programming training and experience. Each group was composed of 15 participants. We sent an e-mail to a mailing list of students of the Department of Control and Computer Engineering of our university (POLITO) to recruit participants with programming experience. For the second group of participants, we held a brief seminar during a psychology class of the University of Turin (UNITO), to introduce the students to IoT and trigger-action programming. At the end of the seminar, we explained the nature of the study that we wanted to carry out, and we recruited 15 volunteers.

During the study, we gave the participants an initial questionnaire to gather general information. Table 3.4 summarizes the demographics of our participants. All the participants were students (15 female) with a mean age of 22.23 years ($SD = 2.19$). We asked them about their programming experience, their experience with connected entities, and whether they were familiar with IFTTT, through three questions based on a Likert scale from 1 (Very low) to 5 (Very high). The 15 participants with formal programming training indicated a quite high programming experience level ($M = 3.33$, $SD = 0.62$). The difference with the other group of participants was substantial ($M = 1.13$, $SD = 0.35$). Even the experience with smart devices and online services was different between the two groups ($M = 2.73$, $SD = 0.70$ for the POLITO students, and $M = 1.73$, $SD = 1.16$ for the UNITO students). Instead, we found that the declared experience with IFTTT was similarly low for both groups ($M = 1.47$, $SD = 0.74$ for the POLITO students, and $M = 1.00$, $SD = 0$ for the UNITO students).

Table 3.4: General demographics in the *EUPont* user study.

Id	Age	Gender	University	Programming Experience	IoT Experience	IFTTT Experience
P1	25	M	POLITO	3	2	1
P2	19	M	POLITO	3	2	1
P3	22	M	POLITO	2	3	1
P4	21	F	POLITO	3	2	1
P5	26	M	POLITO	4	3	3
P6	24	M	POLITO	4	2	1
P7	25	M	POLITO	4	3	1
P8	22	M	POLITO	4	4	3
P9	24	F	POLITO	3	3	2
P10	25	M	POLITO	4	2	1
P11	25	M	POLITO	3	2	1
P12	25	M	POLITO	4	3	2
P13	24	M	POLITO	3	3	2
P14	24	M	POLITO	3	4	1
P15	23	M	POLITO	3	3	1
P16	20	F	UNITO	1	1	1
P17	20	F	UNITO	1	1	1
P18	22	M	UNITO	1	1	1
P19	24	F	UNITO	2	2	1
P20	19	F	UNITO	2	3	1
P21	20	F	UNITO	1	1	1
P22	19	F	UNITO	1	5	1
P23	20	F	UNITO	1	1	1
P24	20	F	UNITO	1	3	1
P25	23	F	UNITO	1	2	1
P26	20	F	UNITO	1	2	1
P27	20	F	UNITO	1	1	1
P28	23	M	UNITO	1	1	1
P29	21	F	UNITO	1	1	1
P30	22	F	UNITO	1	1	1

Procedure and Tasks

We gave participants the initial questionnaire and a privacy module. Then, we introduced them to trigger-action programming and to the IFTTT environment. In this "training" phase, we showed to the participants each connected entity involved in the study, and we defined an IF-THEN rule in both the *IFTTT-like* and *EUPont* interfaces as an example. After the training phase, participants started to complete the five tasks. The order of the tasks and the order of the used iterfaces were counterbalanced. At the end of each session, we performed a final debriefing with the participant, with the aim of finding the perceived advantages and disadvantages of the two experimental representations. Task descriptions, questionnaires, debriefing questions and answers, and moderator interventions were in Italian, and translated

for the purpose of this thesis.

We designed five tasks of the same type to be completed with both the *IFTTT-like* and the *EUPont* interface. Each task consisted of two different parts: a *user scenario* and a *goal*. The *user scenario* described a generic user, the devices she owned and her registered services, and some of her typical activities. It did not contain any explicit reference to the general categories of the *EUPont* ontology. Moreover, to avoid any biases towards ML and HL triggers and actions, the user scenario reflected the contemporary low-level abstraction by specifying all mentioned connected entities in "low-level" terms, i.e., with manufacturers and brands. The *goal* defined a specific behavior that the user wanted to obtain from her connected entities, and it was definable with one or more IF-THEN rules. The tasks were:

T1 **User scenario:** Mary is a researcher in a university. She is environmentally friendly, and, in particular, she is interested in saving energy. However, she is distracted, and she often forgets to turn the lights off. For this reason, she started to gather information about IoT devices, and she equipped her home with some smart lights. She installed two Philips Hue lamps in her bedroom, and two Stack Lighting lamps in the living room and in the kitchen. Furthermore, she used a Samsung SmartThings Hub to remotely control the doors and the surveillance system. Also her office is equipped with smart devices: a surveillance system connected to a SmartThings hub, and few LIFX smart lights.
Goal: Mary would like to automatically turn the lights off when she leaves a room or her office.

T2 **User scenario:** John lives in the countryside, near Turin. He loves sport, and, in particular, cycling. When available, he always uses bike-sharing services. John reaches his workplace, an engineering study with offices in Turin and Milan, by train. When he arrives at the train stations of Turin or Milan, he checks the availability of bikes with the bike-sharing services of the 2 cities. If there are not available bikes, he has to go to work on foot, thus arriving late, typically. When this happens, John alerts his manager with a phone call from your iPhone.
Goal: When the train is approaching a station, John would like a bike to be automatically booked.

T3 **User scenario:** The mother of Jack is very thoughtful, and she is always worried when her son goes around alone. In particular, she is anxious when Jack takes the bus, the subway, or a friend's car, and she would like to constantly receive updates from Jack on her iPhone. Unfortunately, Jack always forgets to warn his mother when he arrives at his destination.

Goal: When he uses a means of transport and he arrives at his destination, Enrico would like to automatically send a message to his mother from his Android smartphone.

T4 **User scenario:** Paul is an architect that lives in Turin. He loves technology, and he has equipped his home with a Nest thermostat, that he can control with his Android smartphone, to regulate the temperature of all his rooms. Paul is very satisfied, because he realized that he can save money with heating automation. For this reason, he decided to equip his office with a Netatmo thermostat.
Goal: Paul is always cold, and he would like to automatically set the temperature to 22 Celsius degrees when he enters an indoor space.

T5 **User scenario:** Mark and Andrew are managers of an important tech-company with offices in Turin, Milan, and Rome. The offices are equipped with many IoT technologies: doors are connected to a SmartThings hub, and there are Nest smart cameras and Samsung air conditioners in each room. When Mark and Andrew meet, they typically have a coffee and discuss their work plans for the near future. Both Mark and Andrew are constantly moving between the various company offices, and they find it difficult to meet each other.
Goal: Mark would like Andrew to be automatically notified on his iPhone when they are in the same company office.

To investigate whether the participants were aware of the potential and the limitations of the two representations, tasks were divided in a) solvable (completely or at least approximately) in both the interfaces (T1, T2, and T4), or b) impossible to be solved in the *IFTTT-like* interface (T3 and T5). One task for each category (T1 and T5) was performed by the participants following the think-aloud protocol. Users could complete a task with one or more IF-THEN rules, or, in any moment, they could mark a task as impossible if they thought the task was not completely realizable. For example, proximity of other people (e.g., in T5) can be included in *EUPont* rules, but it is not supported by contemporary low-level representations such as the one adopted in IFTTT. During the study design phase, we defined a possible solution for each task in both representations. In the reported example, the task could be solvable with one rule in the *EUPont* interface:

- if I leave an indoor place, then turn the lights off in the same indoor place.

With the *IFTTT-like* interface, instead, the task could be successfully completed with the following set of rules:

- if the SmartThings hub no longer detects presence (from the bedroom camera), then turn the bedroom Philips Hue lamps off;

- if the SmartThings hub no longer detects presence (from the living room camera), then turn the living room Stack Lighting lamp off;
- if the SmartThings hub no longer detects presence (from the kitchen camera), then turn the kitchen Stack Lighting lamp off;
- if the office SmartThings hub no longer detects presence (from the office camera), then turn the office LIFX lamps off.

Measures

Data from the study depended on two main factors (independent variables): the interface used to carry out a task (*IFTTT-like* or *EUPont*), and the participants group (users with or without programming experience).

For each task completion (with both interfaces), we collected the following *quantitative measures*: a) the time (in seconds) needed by the participants to complete a task[10], b) the number of incorrect triggers, c) the number of incorrect actions in the inserted rules, and d) the number of times that a participant pressed "back". Since the number of required rules for completing a task differed in the two representations, we normalized the four measures with respect to the number of rules inserted by the user in the given task completion. Then, to conduct statistical analysis, we calculated the means of these measures by considering all the tasks completed by a user in the same representation. At the end, for each user, we obtained the following four dependent variables:

- **incorrect triggers:** the normalized average number of incorrect triggers in rules defined in a given representation;
- **incorrect actions:** the normalized average number of incorrect actions in rules defined in a given representation;
- **back number:** the normalized average number of times a participant pressed "back" in the definition of a rule in a given representation;
- **duration:** the average time (expressed in seconds) needed by the participants to define a trigger-action rule in a given representation.

We also collected *qualitative measures* from the study by observing the users in the two tasks performed with the think-aloud protocol, and the perceived advantages and disadvantages of the two representations in the final debriefing. Furthermore, we analyzed whether participants recognized impossible tasks.

[10] except for think-aloud tasks

3.1.5 Results

EUPont Effectiveness

To understand whether the *EUPont* representation *effectively* helps and guides users in defining their IF-THEN rules (**RQ3**), we analyzed the effect of the interface independent variable (*IFTTT-like* or *EUPont*) over the four dependent variables. Table 3.5 reports the means of the four analyzed measures in both interfaces.

Table 3.5: Means and standard deviations of the four normalized dependent variables investigated in the study.

	IFTTT-like	***EUPont-like***
Incorrect Triggers	0.279 ± 0.028	0.120 ± 0.025
Incorrect Actions	0.203 ± 0.028	0.038 ± 0.010
Back Number	0.971 ± 0.150	0.588 ± 0.117
Duration	$39.647s \pm 2.600s$	$25.054s \pm 1.948s$

We conducted four different two-way mixed ANOVA in SPSS with a post-hoc analysis with Bonferroni correction. We considered incorrect triggers, incorrect actions, the number of back, and the tasks' duration as the dependent variables; the used interface as the within-subject; and the participants group as the between-subject. The Mauchly's sphericity test was satisfied for the used representation in all the four analysis.

a) We found that, if we ignore whether the participant had programming experience or not, the number of errors in the selection of triggers for rules defined with different representations significantly differ ($F(1,28) = 19.14$, $p < .05$): on average, participants defined *EUPont* rules with less errors during the definition of triggers with respect to *IFTTT-like* rules (0.120 ± 0.025 vs. 0.279 ± 0.028, respectively). A post-hoc test with the Bonferroni correction revealed that this difference was statistically significant ($p < .05$). The programming experience level of the participants did not significantly influence the variable ($F(1,28) = 3.347$, $p > .05$). Furthermore, there was not a significant interaction between the used interface and the programming experience in terms of incorrect triggers ($F(1,28) = 8.348 \times 10^{-6}$, $p > .05$).

b) We found that there was a significant main effect of the interface used ($F(1,28) = 35.837$, $p < .05$). As for the triggers, by ignoring the programming experience, the number of errors in the definition of the actions for rules defined with different representations significantly differs. Participants made less errors during the definition of actions in *EUPont* rules, as the means of the incorrect actions variable were higher with the *IFTTT-like* than with the *EUPont*

interface (0.203 ± 0.028 vs. 0.038 ± 0.010, respectively). Also in this case, a post-hoc test with Bonferroni revealed that this difference was statistically significant ($p < .05$). The programming experience level of the participants did not significantly influence the incorrect actions variable ($F(1,28) = 0.001$, $p > .05$) nor there was a significant interaction between the interface and the programming experience in terms of incorrect actions ($F(1,28) = 0.343$, $p > .05$).

c) We found that the average time needed for defining rules was significantly different for the two iterfaces ($F(1,28) = 25.402$, $p < .05$), independent from the programming experience. Users defined rules with the *EUPont* representation faster than with the *IFTTT-like* representation. In fact, the means of the duration variable were lower for the *EUPont* than for the *IFTTT-like* interface ($25.054s \pm 1.948s$ vs. $39.647s \pm 2.600s$), as confirmed by a post-hoc test with the Bonferroni correction ($p < .05$). Also in this case, there was not a significant effect of the programming experience level of the participants on the duration variable ($F(1,28) = 0.263$, $p > .05$) nor a significant interaction between the used interface and the programming experience in terms of duration ($F(1,28) = 0.354$, $p > .05$).

d) We found that there was not a significant main effect of the used interface on the back number variable ($F(1,28) = 4.152$, $p > .05$). However, the back number was lower with the *EUPont* interface (0.588 ± 0.117 vs. 0.971 ± 0.150).

Users' Perception

We analyzed the qualitative data collected during the study to establish which representation is preferred by the users, and what are the perceived advantages and disadvantages of the two solutions (**RQ4**). The qualitative analysis was conducted by two researchers in an iterative coding process. Inter-rater reliability was determined using Fleiss'Kappa coefficient. First, a researcher transcribed the experiment videos and interviews. Then, both researchers individually coded the transcriptions. After this phase, they met and discussed disagreements. Eventually, they settled on three code sets: (1) understanding low level limits, (2) avoiding mistakes and confusion, and (3) advantages and disadvantages.

Understanding Low-Level Limits. The limits of the low-level of abstraction provided by the *IFTTT-like* representation were easily recognized by the majority of the participants. During a task resolution, 15 of them asserted that a proximity event of two people was impossible to define with the *IFTTT-like* interface, since the rules definable with IFTTT cannot explicitly involve other users than the rule creator. A user said "*I cannot know the position of another person in this interface.*" All of these 15 participants correctly stated

that the task was impossible to complete. Another 5 participants defined the event anyway, but they explicitly acknowledged they were aware of the approximation they made. In another task, 11 participants were upset because they had to insert the same rule for different technologies and places. Even if the action was the same (turn the lights off), they had to consider all the different lights manufacturers, e.g., a participant said "*I would like to use the same action for all the rules.*"

Avoiding Mistakes and Confusion. From the analysis of the thoughts and behaviors of the participants, it emerges that the *EUPont* interface helped users avoid mistakes in the rules definition. During the solution of a task with the *IFTTT-like* interface, a participant said "*It is important to stay focused, otherwise it's easy to make mistakes and forget something.*" Furthermore, when we looked at the task solutions with the *IFTTT-like* interface, we noticed that errors and omissions were very common, even if users followed a correct reasoning process. This was evident when participants had to face many different technologies, as in the task reported in subsection 3.1.4. In this case, 11 participants forgot to replicate the same IF-THEN rule for all the different devices and rooms, or they incorrectly defined some triggers or actions details (e.g., the specific light type). With more general attributes, the *EUPont* representation seemed to mitigate this problem, since 26 participants completed the tasks without any difficulty and correctly with the related interface. The video recordings analysis shows that users seemed to be more confused in defining IF-THEN rules with the *IFTTT-like* interface. With such an interface, on a total of 60 think-aloud task resolutions, only 2 participants successfully completed a task by immediately identifying the correct triggers and actions (3.3%). In all the other cases, participants changed their mind several times before reaching a solution. On the contrary, 29 think-aloud tasks were completed with the *EUPont* interface without changing idea (48.3%).

Advantages and Disadvantages. By analyzing the data from the debriefings, we coded the feedback given by the participants about the *IFTTT-like* interface in some disadvantages and advantages. First of all, participants acknowledged that a user has to know in advance the devices she owns to define her IF-THEN rules. This disadvantage was especially cited by participants without programming experience. Starting from this fact, tasks in the *IFTTT-like* interface required various (complex) rules, and the required time to define rules was higher than the time required with the *EUPont* interface. The majority of the participants agreed that generic concepts were impossible to define with the *IFTTT-like* interface. Other users were disappointed about the large number of supported connected entities, often perceived as

not necessary. Not surprisingly, the specific nature of the *IFTTT-like* interface emerged as an advantage for some tasks, especially from participants with programming experience, since it allows them to choose the best solution for their needs (e.g., specify the GPS position of the smartphone).

3.2 Discussion and Guidance for Future Research

In this chapter, we explored a new way of personalizing smart devices and services through the definition of abstract and technology-independent IF-THEN rules. The "low-level" representation adopted by contemporary trigger-action programming platforms (e.g., IFTTT) is, in fact, not suitable to overcome the issues brought on by the steady growth of the IoT ecosystem (*low-level abstraction* issue). The fact that devices and services with similar capabilities but different brands are treated as separate entities, and that contemporary solutions only work with well-known devices and services are two evident examples.

To take a step towards a higher level of abstraction, we formally defined *EUPont*, an ontological high-level representation for end-user development in the IoT. The model allows the definition of abstract and technology-independent IF-THEN rules, e.g., *"if I enter a closed space, then cool the environment,"* and it supports the selection of *currently available* real devices and services *able* to reproduce the defined abstract behaviors. By translating a large set of IFTTT rules in the *EUPont* representation, we demonstrated that the model is more expressive than the representation models offered by contemporary trigger-action programming platforms. Furthermore, thanks to a user study with 30 participants, we successfully demonstrated that the *EUPont* representation allows end users to avoid errors and to reduce the time needed to define their IF-THEN rules. Furthermore, it introduces numerous benefits in terms of understandability and ease of use.

Different insights to inspire future research in the fields of IoT, EUD, and trigger-action programming can be extracted from the presented results. We organized them across different themes, ranging from the design of user interfaces to trustfulness, security, and privacy.

Reducing Information and Adapting Contexts. In the upcoming IoT ecosystem, smart devices and online services will not always be knowable a priori and the complexity of the entire IoT ecosystem will continuously increase. As some participants in our user evaluation found, a low-level representation risks the generation of user interfaces that are cluttered and with too much information. By presenting a lower amount of information, the high-level interface powered by *EUPont* allowed participants to define rules in less time, with less errors, and with more guidance towards the choices they had.

During the *EUPont* study, we noticed that some participants often forgot to replicate the same rule for all their available devices in the low-level representation, or they incorrectly defined some specific trigger or action details. By analyzing the experiment videos and interviews, we also found that often users would like to reuse the same Low-Level trigger or action for different rules, since their final meaning was the same. For this purpose, *EUPont* naturally provides ways to adapt trigger-action rules to different contextual-situation, with the aim of addressing extremely contextualized user needs [60]. Clearly, the usage of a higher level of abstraction leads to different aspects that need to be addressed in future works. Abstract behaviors such as "illuminate a place" or "send a message," for instance, could be potentially reproduced in different ways, on the basis of the current context. How can we decide how to reproduce them? Which is the "best" solution for a given user? Different solutions could be adopted, ranging from preference-based approaches to fully automated solutions, e.g., by means of machine learning algorithms.

Custom Level of Abstraction. Even by being able to reproduce abstract behaviors on real devices and services, our results demonstrate that the sole usage of a high-level of abstraction is not sufficient. As some participants of our study noted, the interface for defining IF-THEN rules sometimes appeared to be too generic and did not offer the possibility to manage particular details of involved connected entities, especially in case of HL triggers. Although this can be seen as a normal effect of moving from a more specific model to a more abstract representation, some participants would like to provide more details during the creation of their own rules (e.g., select all the lights but not the shades for illuminating a place). This may suggest that users did not immediately understand the full potential of the High-Level of abstraction of *EUPont* or that they would like more precise control. However, they desired not to have the same amount of details as in the low-level representation. Therefore, a promising direction for future works is to explore multi-level interfaces exposing the hierarchy of possible triggers and actions, ranging from the highest level of abstraction to triggers and actions with more specific details. For example, the Medium-Level of abstraction of *EUPont* could provide more fine-grained control than the highest level of abstraction also for the triggers, e.g., by allowing users to specify how to capture the event of entering their home. The need of more than one level of abstraction is also motivated by the study Ur et al. [136], in which the authors explored the trigger-action paradigm in the smart home scenario. In particular, consistently with other related work [47, 135], they found that participants tended not to mention sensors directly, and they discovered that many participants express triggers at one level of abstraction higher.

Programming by Functionality. The *leitmotif* of the presented work is the

need to put the user at the center of the interaction, so that she can express her needs and desires without recurring to a device-centric or app-centric language, but directly indicating the *functionality* in which she is interested. *EUPont* was designed to model triggers and actions on the basis of the behavior they aim to reproduce, without the need of specifying any technological (e.g., brands or manufacturers) details. In this way, different low-level triggers or actions collapsed in the corresponding ML/HL trigger or action, e.g., all the lamps offered the same `turn lights on` action. The advantages of this modeling pattern have been confirmed by the result of the user study: besides reducing the displayed information, a different organization of triggers and actions, i.e., in terms of their final functionality, helped participants defining their IF-THEN rules in terms of effectiveness and efficiency. Different approaches could be investigated in future works to further explore the benefits of "programming by functionality" in trigger-action programming platforms. Chapter 4 of this thesis is an example: through optimization methods and recommender systems, trigger-action programming platforms could provide users with more support to *discover* functionality. Another possibility could be the design of conversational interfaces, through which users could be guided in exploring the range of different functionality offered by their connected entities.

Trustfulness, Security, and Privacy. Some participants in our user study highlighted that they should have a profound *trust* in a system adopting *EUPont*. Similarly, two participants were worried about privacy and security issues that a system based on the high-level model could present, especially due to the abstract nature of the representation. To adopt a high-level representation model such as *EUPont*, future works would need to carefully consider trustfulness, privacy, and security issues. This may include warning mechanisms in user interfaces to alert users about possible dangerous rules, and debug features to help people simulate and foresee rule behavior under different conditions [46]. In our work, we started to investigate end-user debugging approaches for contemporary trigger-action platforms (Chapter 5). Debug could be even more important in case of abstract IF-THEN rules: based on the context, showing on which real devices and services a high-level rule is mapped onto could increase the system trustfulness.

+

Chapter 4

Discovering IF-THEN Rules and Functionality

With connected entities modeled on the basis of the underlying brand or manufacturer, the number of possible combinations among triggers and actions of different technologies in contemporary trigger-action programming platforms is high, and the number of shared rules is growing (*information overload* issue). IFTTT and Zapier, for example, force users to define their rules by searching between 500 and 1,000 supported entities displayed with a meaningless order, each one with its own triggers and actions, while the number of publicly available rules on IFTTT already exceeded 200,000 back in September, 2016 [137]. Consequently, a user who wants to customize the behavior of her smart home through IFTTT has many possibilities: she can define a temperature to be set on her *Nest* thermostat whenever her *BMW* smart car is approaching the home area, or she can make the *Philips Hue* lamp in the kitchen turn on whenever the *Arlo* security camera detects some movements. Even in such a limited scenario, the 4 mentioned technologies offer 15 triggers and 19 actions on IFTTT, thus generating 285 candidate rules. This number is even larger if we consider specific details of each trigger and action, such as the temperature threshold for the thermostat or the light intensity of the lamp.

This particular type of information overload suggests the need of providing users with more support for discovering and managing rules and related functionality [137]. Therefore, instead of using *EUPont* for *defining* rules with a new level of abstraction, we explored how to use the model to help users *discover* proper "low-level" triggers, actions, and rules without requiring any radical change in the adopted representation model. To this end, we proposed:

a) *EUDoptimizer* (Section 4.1), an optimization approach to dynamically redesign layouts in trigger-action programming interfaces in an interactive way, i.e., by considering the choices made by end users during the rule definition process; and,

b) *RecRules* a hybrid and semantic recommendation system of IF-THEN rules.

Part of the work described in this chapter has been previously published in three different papers. The description and the evaluation of *EUDoptimizer*, in particular, is based on the work published in [102] and [34], while *RecRules* was initially presented in [36].

4.1 *EUDoptimizer*: Defining IF-THEN Rules with an Optimizer in the Loop

As already explained in Section 1.1.1, contemporary trigger-action programming platforms force users to define IF-THEN rules through a multi-step procedure, i.e., what we called the rule definition process. The more critical step, in particular, is the Connected Entity Selection, i.e., the selection of the generic connected entity to be used in the trigger and in the action. Indeed, user interfaces of platforms like IFTTT and Zapier force users to browse thousands of supported entities through unordered grid layouts, i.e., a particular type of menu where items are organized in rows and columns, without any particular support. To better assist users in defining rules, we designed and implemented *EUDoptimizer* (***E**nd-**U**ser **D**evelopment **optimizer***), an optimizer in the loop to *dynamically* redesigning such grid layouts in an *interactive* way, i.e., by considering the choices made by end users during the rule definition process. The goal is to promote the discovery of the "right" connected entity to be used for defining the trigger or the action, according to the user's need. To reach our goal, we defined two different predictive models to be used in a multi-objective optimization problem. In particular, we adapted Search-Decision-Pointing (SDP), a state-of-the-art predictive model of user performance in linear menu search, to work with grid layouts. Furthermore, we proposed the Functionality Similarity Model (FSM), a novel model based on *EUPont* to take into account whether different and heterogeneous technologies provide similar functionality. In line with previous works [137], we claim that users would benefit from more support in discovering functionality, rather than being forced to get acquainted with technological details. As demonstrated in Chapter 3, indeed, users are often interested in what a device or service can do, and they reason about abstract behaviors, e.g., *"turn on the lights of the kitchen"*, rather than specific technologies, e.g., *"turn on the Philips Hue lamp in the kitchen."*

To explore the design space of grid-based trigger-action programming interfaces, we considered two different optimization algorithms, i.e., Simulated Annealing and Ant Colony Optimization, and we integrated those optimization methods on top of IFTTT.

4.1.1 Background

With *EUDoptimizer*, we adopted combinatorial optimization methods for redesigning trigger-action programming user interfaces. Applying optimization methods to user interface design is a long standing topic in human-computer interaction research: when assumptions are appropriate, optimization methods offer a greater-than-zero chance of finding an optimal design [112]. Optimization methods have been firstly adopted for keyboard layouts [153, 113], and then in many other application areas, e.g., accessibility [55], menus [94, 7], sketching [134], and web layouts [123]. Since trigger-action programming user interfaces are typically organized through grid menus, our work is strictly related to the menu optimization problem.

Due to large design space, menus are good candidates for optimization problems. A menu with n elements, in fact, can be organized in $n!$ ways. Design heuristics, e.g., displaying frequently used items at the top, may be effective for small n but fail with larger n or if additional human factors such as semantic relationships among items are considered [7]. Combinatorial optimization methods, instead, explore a large number of designs in order to find a good (preferably optimal) solution that minimizes or maximizes an objective function. While the computational cost is often a problem, reasonable solutions can be obtained by adopting *interactive* approaches, i.e., by involving users in redefining and refining the optimization problem [7]. In *EUDoptimizer*, the problem of defining trigger-action rules contains steps that are intrinsically interactive: components layout for defining an action, for example, may change on the basis of the defined trigger.

Similarly to previous works, we followed a model-based optimization approach [112]. Unlike heuristic approaches, which do not predict effects on end users, model-based optimization exploits predictive models of user performance and layout perception. The idea is to represent a design problem, along with a design knowledge, as an objective function. Then, a search algorithm is used to iteratively improve designs for the stated object. Several models of user performance and layout perception have been proposed for specific tasks. Examples can be seen in Sketchplore [134], a multi-touch sketching tool that uses a real-time layout optimizer, and MenuOptimizer [7], an interactive design tool for menus that exploits the SDP [26] model, and a model of expected item groupings. Similarly to MenuOptimizer, we adapted SDP to predict the selection time of a connected entity from a grid layout, and we involved the user in the optimization process: by defining the trigger, users interactively provide the optimizer with fundamental information to produce the layout for defining the action.

4.1.2 Optimizing IF-THEN Rule Definition

Figure 4.1 provides an overview of our optimization approach to define IF-THEN rules. *EUDoptimizer* calculates an optimized layout to be used in the Connected

Entity Selection step of the trigger definition (*optimized trigger layout*, (a)). In the optimized layout, in particular, the optimizer finds a trade-off for a) displaying supported connected entities according to their usage probability, and b) pulling together connected entities that share functionality similarities in logical groups. The user exploits the optimized layout to select the generic connected entity involved in the trigger, e.g., a *Nest* thermostat. As the user is completing the definition of the trigger, e.g., with the selection of a specific event, *EUDoptimizer* dynamically uses the user's choice (b) to refine a precomputed layout to be used in the action definition phase (*intial action layout*, (c)). The optimizer, in particular, finds a trade-off for promoting the entities most commonly associated with *Nest* thermostats, i.e., the generic connected entity selected for the trigger, without affecting groups of functionally similar entities. The user can use the *optimized action layout* (d) to select the connected entity involved in the action.

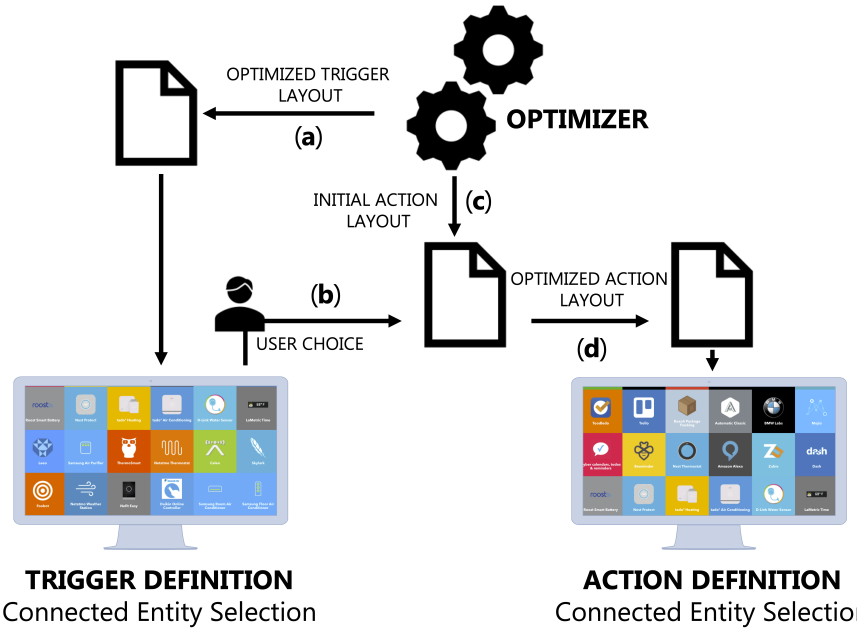


Figure 4.1: An overview of *EUDoptimizer*. The optimizer calculates an optimized layout for defining triggers (a), and it dynamically uses the user's choice (b), along with an initial action layout (c), to produce an optimized layout for defining actions (d).

To further exemplify our optimization approach, let us consider the following scenario of trigger-action programming:

> *John owns many devices always connected to the Internet, and he is subscribed to various social networks and cloud services. John has just*

started to use EUDoptimizer to define trigger-action rules for customizing the joint behavior of his devices and services. John would like to have all the photos taken with his two smartphones saved in different places, e.g., for backup purposes, and for making them available on all his other devices. Furthermore, he would like to automatically share the photos on image sharing platforms. John opens EUDoptimizer and starts to define a trigger-action rule.

Trigger Definition

When John starts to define the trigger, *EUDoptimizer* shows a grid layout in which popular photo and video technologies with similar functionality are grouped together on the top of the menu (as shown later in Figure 4.3), thus facilitating John in finding what he needs. John select the *iOS Photo* entity, that allows him to define a trigger to monitor every time he takes a photo on his iOS smartphone.

Action Definition

The grid menu for selecting the connected entity for the action dynamically change according to the connected entity chosen for the trigger. *EUDoptimizer* already calculated an initial action layout on the basis of usage probabilities and functional similarities. As soon as John selects *iOS Photo* during the definition of the trigger, *EUDoptimizer* starts to refine the initial solution according to the user choice. When John starts to define the action, *EUDoptimizer* shows the grid layout of Figure 4.4. Not surprisingly, the functionality in which John is interested are not uncommon. At the top of the layout, John can find different connected entities that allow him to reach his goals. With *Dropbox*, *Google Drive*, etc., John can save the photos taken with his smartphone on the cloud, thus making them available for all his other Internet-enabled devices. John decides to the define an action for saving the photos on his *Dropbox* folder. John is very satisfied of *EUDoptimizer*: now he can define many other rules to customize his other smartphone, an Android-based device, and to save and share his photos with many different online services.

4.1.3 SDP-FSM: A Predictive Model for Trigger-Action Programming

The goal of *EUDoptimizer* is to employ model-based optimization methods to *dynamically* redesign grid layouts in trigger-action programming interfaces in an *interactive* way, i.e., by considering the choices made by end users. Model-based optimization methods need to be supported by valid and comprehensive predictive models. One of the most recent model of menu performance is SDP [26]. We adapted such a model to work with grid layouts in trigger-action programming

interfaces displaying connected entities. We used SDP in combination with a novel model that takes into account the *expectation* of item groups, i.e., the expectations of users in finding certain items together. We called our model for item grouping Functionality Similarity Model (FSM). It considers similarities between connected entities in terms of the functionality they allow to define through their triggers and actions. Roughly speaking, a *Philips Hue* lamp shares some functionality with a *Hunter Duglas* blind for defining an action, because they both allow the increase or decrease of the brightness of a room. The *Android Location* service and the *Nest* surveillance camera, instead, allow the definition of triggers with the same final goal, i.e., to monitor when someone is entering a place. To discover such similarities, we used the Semantic Web framework, and, in particular, the *EUPont* ontology.

SDP: Search Decision Pointing

SDP is a state-of-the-art model of human performance in linear menu search. It incorporates both Hick-Hyman and Fitts' law, and integrates a transition from novice to expert performance. We adapted the model to be used for grid layouts, i.e., a particular type of menu, by using the euclidean distance between items.

SDP predicts the average performance of a menu through the following formula:

$$SDP = \sum_{i=1}^{n} p_i \cdot T_i, \tag{4.1}$$

where T_i is the selection time of an item i from the menu, p_i is the probability of item i being selected, i.e., its usage probability, and n is the number of menu items.

Item Probabilities. In our work, we preliminary define the probability function p_i as the *frequency probability*, i.e., the probability of a generic connected entity i to be used in a trigger or in an action. The idea is to move towards the top of the grid layout the entities most commonly selected as triggers or actions. When the user selects a generic entity k for the trigger, the SDP model interactively changes in

$$SDP_k = \sum_{j=1}^{n} p_{kj} \cdot T_j, \tag{4.2}$$

where p is reformulated as the *bigram probability* p_{kj} of selecting an action entity j after having selected the entity k for the trigger. The idea is to move towards the top of the grid layout the connected entities most commonly connected to the entity k used for defining the trigger.

Selection time. For predicting the selection time T_i of an item i from a menu, SDP uses the following formula:

$$T_i = (1 - e_i) \cdot T_{st} + e_i \cdot T_{dt} + T_{pt}, \tag{4.3}$$

Consistently with previous work [83, 20], we assume a predominantly top-to-bottom search order for end users, i.e., novices, that want to define trigger-action rules. The parameters we use in the formula are the same used both in the original paper [26] and in the optimizations carried out in MenuOptimizer [7]. Parameters are therefore calibrated according to the analysis of two opposite menu designs, i.e., static and random menus [26]. More specifically:

- e_i models the user expertise with the item i, and varies between 0 (inexpert) and 1 (expert) according to the formula $e_i = L \cdot (1 - 1/t_i)$. In the formula, t_i represents the number of previous selections of the item i, while L represents the "learnability" of the menu, with $L = 0$ that is used for not learnable menus, e.g., with items that move randomly, and $L = 1$ that is used for entirely learnable menus, e.g., when items do not change their position over time. We initially suppose that users do not know anything about the used menu, i.e., $e_i = 0$ for all the menu items. Furthermore, we differentiate the value of L on the basis of the used menu. For layouts that rarely change over time, e.g., all the grid menus of contemporary trigger-action programming platforms, or the optimized trigger layout of Figure 4.1, we set $L = 1$. Instead, we set $L = 0.5$ for the optimized action layout of Figure 4.1. Here, in fact, the layout may vary depending on the defined trigger, but the optimizer continues to maintain logical groups of functionally-related entities.

- $T_{st} = b_{st} \cdot n + a_{st}$ is the **search time**, i.e., the time to localize the item in the menu, linear with the total number of items n when the user is inexperienced. In the formula, the parameters $b_{st} = 0.08$ and $a_{st} = 0.3$ are constants.

- $T_{dt} = b_{dt} \cdot \log_2(1/p_i) + a_{dt}$ is the **decision time**, i.e., the time to decide from among items, given by the Hick-Hyman law once the user becomes expert with the item i. The factor $\log_2(1/p_i)$ represents the entropy of the item, and it is based on its frequency (or bigram) probability. The parameters $b_{dt} = 0.08$ and $a_{dt} = 0.24$, instead, are constants.

- $T_{pt} = a + b \cdot \log_2(A_i/W_i + 1)$ is the **pointing time**, i.e., the time to "point" to the item, described according to the Fitts' law, which predicts that items closer to the top are faster to select. A_i and W_i are the amplitude of movement to the target item i and its width, respectively, while $a = 0.37$ and $b = 0.13$ are constants. In our work, we set W_i as the (fixed) width of the boxes displaying connected entities. A_i is instead modeled as the euclidean distance from the top left corner of the grid menu to the target item i.

FSM: Functionality Similarity Model

We defined the Functionality Similarity Model to allow *EUDoptimizer* to produce groups of connected entities that are *functionally* correlated, even if they are

heterogeneous technologies. By considering the *EUPont* categorization of triggers and actions (Section 3.1.2), the Trigger Functionality Association (FA_t) between two generic connected entities i and j is calculated as:

$$FA_t(i,j) = \alpha_f \cdot LL_t(i,j) + \beta_f \cdot ML_t(i,j) + \gamma_f \cdot HL_t(i,j), \tag{4.4}$$

where:

- $LL_t(i,j)$ is the number of Low-Level (i.e., IFTTT-like) triggers shared by i and j.
- $ML_t(i,j)$ is the number of Medium-Level triggers shared by i and j.
- $HL_t(i,j)$ is the number of High-Level triggers shared by i and j.
- α_f, β_f, and γ_f sum to 1, and are used to weights the three elements modeled in FSM, i.e., LL_t, ML_t, and HL_t.

In the same way, the Action Functionality Association (FA_a) between i and j is calculated as:

$$FA_a(i,j) = \alpha_f \cdot LL_a(i,j) + \beta_f \cdot ML_a(i,j) + \gamma_f \cdot HL_a(i,j) \tag{4.5}$$

Given the hierarchical characterization of triggers and actions in *EUPont*, connected entities that shares Low-Level (very specific) triggers and actions are intrinsically more similar, in terms of functionality, than connected entities that only shares more abstract Medium or High-Level triggers and actions. For this reason, we set $\alpha_f = 0.6$, $\beta_f = 0.3$, and $\gamma_f = 0.1$, with the aim of promoting the creation of groups of connected entities that are strongly characterized by similar functionality.

To use the model in a minimization problem, we exploited the pairwise Functionality Associations to compute the Functionality Incoherence score of a given grid menu, both for the definition of triggers and actions (FI_t and FI_a):

$$FI_t = \sum_{i=1}^{n} \sum_{j=i+1}^{n} FA_t(i,j) \cdot d(i,j), \tag{4.6}$$

$$FI_a = \sum_{i=1}^{n} \sum_{j=i+1}^{n} FA_a(i,j) \cdot d(i,j), \tag{4.7}$$

where $d(i,j)$ is the euclidean distance between objects i and j in the grid layout. As shown in Figure 4.3c, the FSM model has desirable effects on the optimizer: connected entities that offer triggers (or actions) with similar functionality tend to be pulled together, while unrelated entities are moved away.

4.1.4 Optimization Problem and Methods

Problem Formulation

To explore the design space looking for "good" or "desirable" grid menu alternatives, we defined a multi-objective task. The goal is to minimize a weighted combination of the outputs of the the two models M_i exploited by *EUDoptimizer*, i.e., SDP and FSM:

$$\text{SDP-FSM} = \min \sum_{i=1}^{2} \lambda_i \cdot M_i, \tag{4.8}$$

where the sum of all the weights λ_i is 1. In our implementation, we empirically tuned the λ values thanks to the trials performed in the technical assesment of the implemented algorithms (Section 4.1.5).

To make the single objectives less sensitive to weight selection, we normalized each M_i with the the objective value θ_i calculated for an initial point x_0:

$$\theta_i = M_i(x_0). \tag{4.9}$$

The problem for designing the grid menu for trigger definition is therefore:

$$\min \left(\lambda_1 \cdot SDP + \lambda_2 \cdot FI_t\right) \tag{4.10}$$

while for the action the problem is:

$$\min \left(\lambda_1 \cdot SDP + \lambda_2 \cdot FI_a\right) \tag{4.11}$$

Based on the interaction of the user, i.e., when the user select a technology k, the problem for the action changes, and becomes:

$$\min \left(\lambda_1 \cdot SDP_k + \lambda_2 \cdot FI_a\right) \tag{4.12}$$

Solving the Optimization Problem

The problem of designing menu systems, both linear and grid-based, can be formulated as a Quadratic Assignment Problem (QAP) [7]. Developed in operational research, QAP [115] allows the modeling of relationships between elements of two sets to minimize the total pairwise cost. In designing menus, m items have to be assigned to m predetermined locations in order to maximize usability, e.g., expected selection time, menu coherence, etc. QAP is a NP-hard problem, and it is considered one of the hardest optimization problems since general instances of size $m > 20$ cannot be solved to optimality. In our case, for example, m technologies (typically with $m > 200$) can be organized in $m!$ ways. Given the complexity of the problem, we cannot claim global optimality, e.g., through exact methods. To attack the problem, we exploit metaheuristic strategies. A heuristic is a technique that seeks near-optimal solutions at a reasonable computational cost without guaranteeing optimality. Some heuristic methods, however, can get easily trapped in a

local optimum: metaheuristics methods modify their use of heuristics methods as optimization progresses [62]. Our implementation, described in the next section, supports two metaheuristics successfully used for QAP problems, i.e., Simulated Annealing [143, 28] and Ant Colony System [48].

Algorithm 1 Simulated Annealing

Let $s = s_0$
for $k = 0$ through k_{max} **do**
    $T \leftarrow \text{temperature}(k/k_{max})$
    Pick a random neighbour, $s_{new} \leftarrow \text{neighbour}(s)$
    if $P(E(s), E(s_{new}), T) \geq \text{random}(0, 1)$ **then**
        $s \leftarrow s_{new}$
Output: the final state s

Algorithm 2 Ant Colony System

while (not converged) **do**
    Position each ant in a starting node
    repeat
        for all ant **do**
            Chose next node with the transition rule
            Apply local pheromone update
    until every ant has built a solution
    Update best solution
    Apply global pheromone update

Simulated Annealing is based on mimicking the metal annealing processing and exploits local and random search in a exploration/exploitation scheme. The main advantage of simulated annealing is its ability to avoid being trapped in local optima. In fact, a neighboring solution is not considered only when it yields to a better objective value: with a certain probability the solution is accepted even if it does not improve the objective. The pseudo-code for the simulated annealing is reported in Algorithm 1. Instead, Ant Colony System is based on the biological metaphor of an ant colony foraging for food, in which multiple searchers cooperate to produce solutions according to a memory of past solutions. The pseudo-code of ACS is presented in Algorithm 2.

4.1.5 Implementation and Technical Assessment

We implemented the proposed optimization approach on top of an IFTTT-like web interface, and we assessed it through different *"off-line"* experiments. While

our approach is generic, i.e., it can be applied to any grid-based EUD interface for trigger-action programming, we chose IFTTT due to the popularity of the platform and the availability of real usage data [137].

Implementation

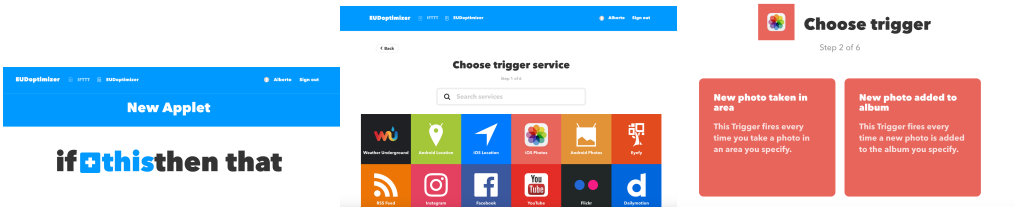
To maintain a high level of interactivity, we implemented a client-server architecture between the optimizer and the user interface for defining trigger-action rules.

Optimizer. We exploited the IFTTT dataset of Ur et al. [137] to calculate the frequency and bigram probabilities to be used in the SDP model. To calculate the Functionality Associations (FA) to be used in the FSM model, instead, we used the translation results described in Section 3.1.3 that linked each IFTTT trigger and action with the corresponding *EUPont* classes. We implemented two different solvers in Python, based on Simulated Annealing (SA) and Ant Colony System (ACS), respectively. We executed them on a regular laptop (a 2015 MacBook Pro with a 2.7 Ghz Intel Core i5 and 8 GB of RAM), separately. To define the algorithm parameters of SA and ACS, we empirically run a set of 100 optimizations by varying the parameter values. Both optimizers provide the same functionality. They initially generate the optimized trigger layout and the initial action layout of Figure 4.1 by using the two "static" versions of the optimization problem (Equation 4.10 and Equation 4.11). Such layouts are periodically recalculated to reflect changes in the probability distributions, e.g., due to new rules defined by the user. As soon as the user selects a connected entity for defining the trigger, each optimizer interactively receives the information. Starting from the initial action layout, each of them starts to explore the problem described by the Equation 4.12 to refine the layout according to the user's choice. When the user finishes the trigger definition, i.e., she completes it with all the required parameters, each optimizer generates optimized action layout, an improved layout version in which the connected entities that are most likely to be used with the selected trigger are promoted towards the top.

User Interface. By exploiting the information extracted from the IFTTT dataset, we modeled a user interface after IFTTT with the AngularJS framework[1]. The interface, shown in Figure 4.2, allows the definition of trigger-action rules exactly as in IFTTT, i.e., by following the multi-step definition procedure described in Section 1.1.1. For defining the trigger, for example, users have to click on the "this" button (Figure 4.2a), and then select the generic connected entity from a grid layout (Figure 4.2b). Finally, they can select the specific trigger to be monitored

[1] https://angularjs.org/, last visited on November 12, 2019

(Figure 4.2c), by filling any required details. To compare *EUDoptimizer* with the original IFTTT, we realized two versions of the same interface, namely *IFTTT* and *EUDoptimizer*. The difference is obviously in the grid menu of Figure 4.2b: while for *EUDoptimizer* the layout of such a menu is provided by the optimizer, in *IFTTT* it reflects the same menu available on the original platform.



(a) New Rule (b) Connected Entity Selection (c) Trigger Selection

Figure 4.2: Some screenshots of the user interface used in the empirical evaluation. The interface resembles IFTTT and allows the definition of trigger-action rules with both the IFTTT version and the *EUDoptimizer* enhanced version. In the IFTTT terminology, rules are named applets, while connected entities are named services.

Technical Assessment

To assess the feasibility of our approach, and to evaluate which optimizer provide better solutions, we carried out different *"off-line"* experiments by changing the number of iterations of the optimization algorithms.

Trigger Definition. We first run *EUDoptimizer* to calculate an optimized trigger layout, thus solving Equation 4.10. Table 4.1 compares the results obtained with 100, 1,000, 5,000, and 10,000 iterations with SA and ACS, respectively. Despite the ACS solver provided better solutions, i.e., lower values of the objective value, for 100 and 1,000 iterations, the SA solver performed better with a higher number of iterations. Furthermore, SA was faster than ACS in all cases. SA, for example, employed 1,031 seconds to produce a solution with 10,000 iterations, while the time employed by ACS with the same number of iterations was considerably higher (20,515 seconds).

Figure 4.3 compares the trigger layout of IFTTT[2] (Figure 4.3a) with two screenshots of the optimized trigger layout calculated in less than 20 minutes by *EUDoptimizer* with 10,000 iterations of SA (Figure 4.3b and Figure 4.3c).

We can observe that:

[2]The figure reflects the grid menu of IFTTT as of September, 2016, i.e., with the data included in the exploited dataset [137].

Table 4.1: Results of Simulated Annealing (SA) and Ant Colony System (ACS) for the optimized trigger layout with 100, 1,000, 5,000, and 10,000 iterations. Comparing to ACS, SA was faster. Furthermore, as the number of iterations increased, SA performed better than ACS, i.e., it produced lower objective values.

	100		**1000**		**5000**		**10,000**	
	SA	**ACS**	**SA**	**ACS**	**SA**	**ACS**	**SA**	**ACS**
Time [s]	**12**	178	**106**	1,896	**536**	10,012	**1,031**	20,515
Obj. value	0.99	**0.89**	0.98	**0.88**	**0.47**	0.86	**0.43**	0.82

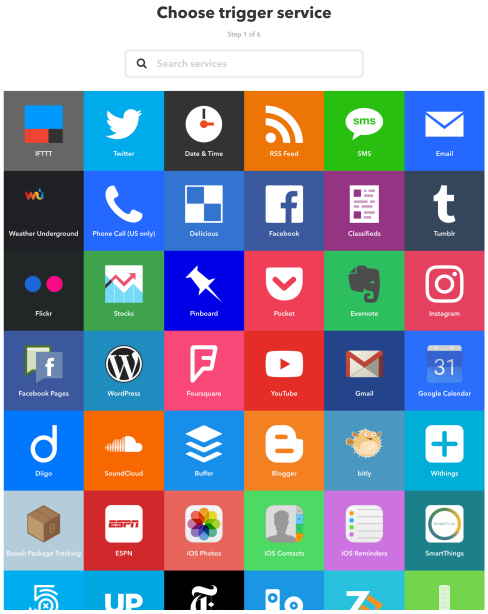
- in the optimized trigger layout, the 10 most frequently used connected entities[3] for defining triggers are prominently placed in the 10 positions closer to the top of the grid menu, as indicated by the added stars of Figure 4.3b. In the IFTTT layout, instead, popular connected entities for trigger definition, e.g., *Android Location* and *iOSPhoto*, are scattered along the menu;
- with respect to IFTTT, where connected entities are displayed in a meaningless order, the optimized trigger layout includes logical groups of connected entities that share functional similarities, e.g., locations (*Android Location*, *iOS Location*) and photos & videos (*iOSPhoto*, *Android Photo*, *Eyefy*, *Youtube*, *Flickr*, *Dailymotion*, and *500px*) in Figure 4.3b. Figure 4.3c further highlights that *EUDoptimizer* is able to pull together entities with functional similarities. The figure, in particular, shows a huge group of hubs, cameras, and doorbells, all related to home security.

By using the otpimized trigger layiut, John, i.e., the user of our scenario of trigger-action programming (Section 4.1.2), can easily find the connected entity for defining his trigger (*iOSPhoto*) at the fourth position of the first row of the menu. In IFTTT, instead, John would find *iOSPhoto* at the third position of the sixth menu row. By using the SDP model (Section 4.1.3), and, in particular, equation 4.3, we can predict how long it will take John to select the *iOSPhoto* item with the 2 layouts. If we assume that John has never seen the menu before, i.e., $e_{iOSPhoto} = 0$, the selection time T_i will include the time to search the item (T_{st}) and the time to point to the item (T_{pt}):

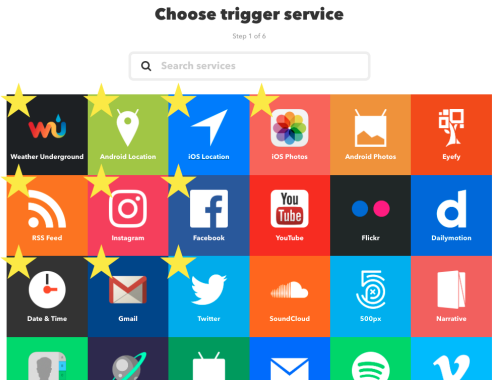
$$T_i = T_{st} + T_{pt} \tag{4.13}$$

Under these circumstances, the time to select *iOSPhoto* is expected to be 3.97 seconds with the IFTTT trigger layout. This predicted time drops to 1.29 seconds by using the optimized trigger layout calculated by *EUDoptimizer*.

[3]according to the dataset of Ur et al. [137]



(a) IFTTT trigger layout



(b) Optimized trigger layout

(c) Optimized trigger layout

Figure 4.3: A comparison between the trigger layout of IFTTT (a) and the optimized trigger layout calculated by *EUDoptimizer* with the Simulated Annealing solver (10,000 iterations, (b) and (c)). With respect to IFTTT, in (b) the 10 most frequently used connected entities are placed towards the top of the menu, as indicated by the added stars. Furthermore, the optimized trigger layout is structured in logical groups of entities with functional similarities: in (c), for example, the yellow border highlights a logical group of entities related to home security.

Action Definition. After calculating the optimized trigger layout, we then run *EUDoptimizer* to calculate an optimized action layout. In this case, each solver

initially calculated (with 10,000 iterations) an initial solution for the menu (initial action layout), i.e., by solving Equation 4.11. Then, we manually fixed the selected trigger entity to *iOS Photo*, and we tested the optimizers to solve the problem defined by Equation 4.12, i.e., the interactive optimization. Table 4.2 compares the results obtained with SA and ACS. Also in this case, SA was faster than ACS, and it performed better than ACS with 5,000 and 10,000 iterations, i.e., it produced lower objective values. Figure 4.4 shows the top part of the best grid menu obtained by *EUDoptimizer* with 10,000 iterations of SA (Figure 4.4). Both the figure and the table confirm that the SA solver produced good solutions in a reasonable amount of time. In fact, 9 of the 10 connected entities most frequently associated[4] with *iOS Photo* are presented on the top of the grid layout. Furthermore, there are logical groups of related entities that allow the definition of similar functionality, e.g., *One Note*, *Nimbus Note*, and *Evernote*.

Table 4.2: Results of Simulated Annealing (SA) and Ant Colony System (ACS) for the optimized action layout with 100, 1,000, 5,000, and 10,000 iterations when the selected trigger entity is *iOS Photo*. As happened for trigger layouts, SA was faster that ACS and performed better as the number of iterations increased, i.e., it produced lower objective values.

	100		**1000**		**5000**		**10,000**	
	SA	**ACS**	**SA**	**ACS**	**SA**	**ACS**	**SA**	**ACS**
Time [s]	**10**	171	**91**	1,669	**468**	11,547	**895**	20,517
Obj. value	1.01	**0.88**	0.98	**0.83**	**0.47**	0.82	**0.45**	0.82

John, i.e., the user of our scenario of trigger-action programming (Section 4.1.2), can find the connected entity to define his desired action (*Dropbox*) in the second position of the first row of the optimized action layout. Online services to store files, indeed, are frequently associated with triggers involving photos & video entities, e.g., *iOS Photo*. According to the SDP model, John can select *Dropbox* in roughly 1.04 seconds, even tough he has never seen the menu before. This time is considerably lower than the one predicted for IFTTT. In the IFTTT action layout, indeed, *Dropbox*, is displayed in the 12th row of the grid menu[5], and the predicted selection time is 6.90 seconds.

[4] according to the dataset of Ur et al. [137]

[5] The *Dropbox* position is calculated by considering the grid menu of IFTTT as of September, 2016, i.e., with the data included in the exploited dataset [137].

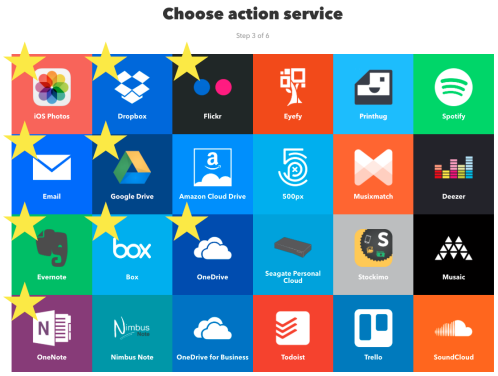


Figure 4.4: Optimized action layout calculated with SA when the selected connected entity in the trigger is *iOS Photo* (10,000 iterations). Stars indicate the most frequently used entities: 9 out of 10 are at the top of the grid layout.

4.1.6 User Evaluation

Thanks to the technical assessment, we selected SA as the solver to be used in *EUDoptimizer*, and we compared the optimized interface with *IFTTT* in a user study with 12 participants, by asking participants to define trigger-action rules with both interfaces. In the study, we explored the following research questions:

RQ5) Does *EUDoptimizer* improve the user performance, i.e., the time needed for defining IF-THEN rules?

RQ6) Does *EUDoptimizer* reduce the cognitive load in the definition of IF-THEN rules with respect to the *IFTTT* interface?

Participants

We recruited 12 participants (4 female and 8 male) with a mean age of 25.91 years ($SD = 4.48$, $range : 19 - 34$). To consider users with and without programming skills, participants were recruited from different background, i.e., Education, Biology, Aerospace Engineering, Management Engineering, and Computer Engineering. 3 participants were undergraduate students, 7 were Ph.D. students, while 2 where post-doc researchers, all coming from different universities. On a Likert scale from 1 (No experience at all) to 5 (I am an expert), participants declared their programming experience level ($M = 3$, $SD = 1.04$), and their experience with IFTTT ($M = 1.67$, $SD = 0.89$).

Procedure & Tasks

We devised a within-subject user study, where we considered the interface version (*IFTTT* vs. *EUDoptimizer*) as the independent variable. We first provided

participants an initial questionnaire to collect demographic data and information about their programming skills and their previous experience with IFTTT. Then, we introduced them to trigger-action programming and to the IFTTT environment, and we explained the nature of the study. During this phase, we showed the interface to the users, by defining an IF-THEN rule in *IFTTT* as an example. After the training phase, we asked participants to complete 6 similar tasks related to the definition of trigger-action rules with both interface versions, without telling them which version they were using. Interface versions and tasks were fully counterbalanced between the participants. The study was carried out in a university office, and took about 30 minutes per participant. All the sessions were audio recorded.

In the study, each task consisted in the definition of a single IF-THEN rule. We defined 6 different tasks that asked participants to replicate trigger-action rules previously extracted from the IFTTT dataset [137]. To explore the full range of possible alternatives, e.g., to evaluate *EUDoptimizer* both with commonly and rarely used connected entities, we first divided the dataset in three layers by grouping together the most common rules (i.e., shared more than 10,000 times), the common rules (i.e., shared 1,000 to 10,000 times), and the uncommon ones (i.e., shared fewer than 1,000 times). Then, we randomly selected 2 rules for each category. The rules, rephrased for the sake of readability, were:

Most common rules

- if the *Weather Underground* service notifies that tomorrow's forecast call for rain, then use *Notifications* to send me a notification;
- if I share a photo on *Instagram*, then add the file on my *Dropbox*.

Common rules

- if I add a photo on *iOS Photo*, then add the file on my *Google Drive*;
- if I receive a new labeled email on *Gmail*, then create a note on *Evernote*.

Uncommon rules

- if the laundry cycle of my *Samsung Washer* is finished, then add an event on *Google Calendar*;
- if the *Nest Cam* recognizes a new sound or motion event, then turn the *Philips Hue* on.

Measures

For each task completion (with both the *IFTTT* and the *EUDoptimizer* interface), we measured the following times:

- **Trigger Time (TT):** the time for selecting the generic connected entity to define the trigger from the grid layout, i.e., the time to complete the trigger Connected Entity Selection step in the rule definition process.
- **Action Time (AT):** the time for selecting the generic connected entity to define the action from the grid layout, i.e., the time to complete the action Connected Entity Selection step in the rule definition process.
- **Rule Time (RT):** the time for defining the entire rule, composed of the Trigger and Action times (TT and AT), and the time needed for completing the other steps of the rule definition process, i.e., Trigger Selection, Trigger Details, Action Selection, and Action Details.

Furthermore, we extracted any consideration made by the participants from the audio registrations.

4.1.7 Results

Improving Users' Performance

All the time measures were lower with the optimized interface. Therefore, *EUDoptimizer* improved the Connected Entity Selection time in the trigger and in the action definition. For defining triggers, users spent 37.29 seconds on average ($SD = 13.44$) to select the connected entity when using the *IFTTT* interface, while they spent 21.69 seconds ($SD = 11.02$) for the same operation when using *EUDoptimizer*. Similarly, users spent 20.87 seconds ($SD = 11.18$) on the *IFTTT* interface to select the connected entity for the action, while they spent 8.26 seconds ($SD = 8.97$), only, for the same operation in *EUDoptmizer*. Such improvements were reflected in the time needed by end users to define an IF-THEN rule (RT): the *EUDoptimizer* interface, in fact, allowed participants to define rules faster that the *IFTTT* interface (51.68 ± 18.28 seconds *vs.* 81.77 ± 23.14 seconds, respectively).

To investigate whether the differences in the measures were significant, we analyzed the effect of the independent variable (*IFTTT* vs. *EUDoptimizer*) over the three dependent variables (TT, AT, and RT) with a one-way repeated measures ANOVA carried out in SPSS. The Mauchly's sphericity test was satisfied in all the three analysis. There was a significant main effect of the used interface on TT ($F(1,11) = 8.30$, $p < .05$), AT ($F(1,11) = 12.46$, $p < .05$) and RT ($F(1,11) = 15.82$, $p < .05$). For all the 3 dependent variables, a post-hoc test with the Bonferroni correction revealed that the mean differences between the two interfaces were statistically significant ($p < .05$), thus confirming the evidence that *EUDoptimizer* improved the selection time in the trigger and action definition phases, and the overall definition time of trigger-action rules (**RQ5**).

To further demonstrate such a statement, we separately analyzed the measures for the two *uncommon rules*, only (Figure 4.6). We were particularly interested in

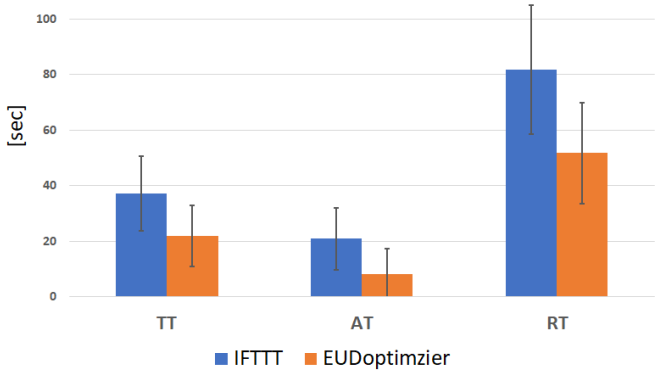


Figure 4.5: Average Trigger Time (TT), Action Time (AT), and Rule Time (RT) compared between the *IFTTT* interface and its *EUDoptmizer* enhanced version. All the time measures are lower in the optimized interface.

evaluating the potential of *EUDoptimizer* in the worst case. In fact, since we used the dataset to weight items by their frequency, connected entities involved in the uncommon rules were *not* placed in the first positions of the produced grid layouts.

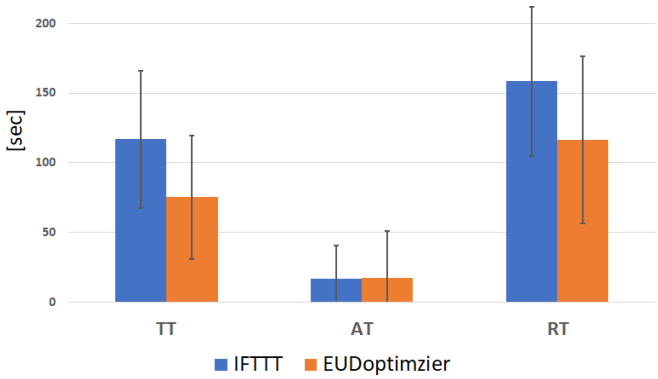


Figure 4.6: Average Trigger Time (TT), Action Time (AT), and Rule Time (RT) compared between the *IFTTT* interface and its *EUDoptmizer* version for the two uncommon rules.

We found that the optimized interface improved the definition of triggers also for the two uncommon rules. The TT measure was lower with the *EUDoptimizer* interface (75.32 ± 44.25 seconds *vs.* 116.81 ± 49.19 seconds). On average, the selection of an entity for defining actions was instead performed with similar performances by participants (16.67 ± 24.21 seconds *vs.* 17.70 ± 33.43 seconds). However, the time for defining the entire rules (RT measure) was considerably lower with the *EUDoptimizer* interface (116.52 ± 59.82 seconds *vs.* 158.31 ± 53.17 seconds).

Reducing Cognitive Load

Qualitative data extracted from the audio recorded files shows that the benefits of *EUDoptimizer* are not restricted to time performance, only. The indications of the participants, in fact, suggest that *EUDoptimizer* reduces the cognitive effort to find connected entities by ordering the components layout with logical groups of *functional-related* elements (**RQ6**). In particular, we found that the majority users were frustrated in using the *IFTTT* interface. Without knowing the differences between the two evaluated interface versions, a participant using the *IFTTT* interface said *"I hate this task, it's very difficult to find the desired technology, I have already looked over the menu 4 times!"* Other 4 participants pointed out that entities seemed to be displayed in a random order, thus making impossible to apply any search criterion. One participant said *"I am forced to search the technology by looking sequentially to all the listed elements, from the top to the end of the grid."* On the contrary, *EUDoptimizer* provided more support for selecting the desired smart devices and online services. 5 participants were happy of the "logical" groups of connected entities showed by *EUDoptimizer*. A participant, for example, said: *"in this interface elements are ordered meaningfully. This helps me to find what I need."* Another participant said *"here the Samsung Washer is near to other appliances of the same type, it's easy to find it."* The other participants were instead happy because the entities they needed, especially for defining actions, were displayed towards the top of the grid layout. A participant said: *"I like this interface [EUDoptimizer] because it proposes me the technologies I need in the first positions and I can immediately select them."*

Despite promising, such findings are preliminary and based on qualitative data, only. Future works would need to confirm the effects of *EUDoptimizer* in reducing cognitive load by means of quantitative measures, e.g., through NASA-TLX surveys.

4.2 Recommending IF-THEN Rules for End-User Development

In parallel with *EUDoptimizer*, we also explored another approach to assist users in discovering and managing rules and related functionality. We tackled, in particular, the emerging problem of recommending rules to end users. To this end, we designed *RecRules*, a hybrid and semantic recommendation system of IF-THEN rules. Recommendation techniques could improve both the *reuse* and the *definition* of trigger-action rules, thus helping users who do not have technological and programming skills to easily customize their smart devices and online services. When browsing rules already created and shared by other users, in fact, a recommender system could suggest relevant rules to be reused. When defining a new rule, instead,

a recommender system could help users to complete their rules, e.g., by dynamically suggesting relevant actions to auto-complete a defined trigger. Through a mixed content and collaborative approach, the goal of *RecRules* is to *recommend by functionality*: it suggests rules on the basis of the final behaviors users would like to define, thus abstracting details such as brands or manufactures.

RecRules is designed as a *top-N* recommendation algorithm and it addresses OWL content-based information and collaborative user preferences in a graph-based setting to train *learning to rank* techniques. By leveraging the *EUPont* model, the algorithm firstly uses a semantic reasoning process to enrich IF-THEN rules with semantic information. Such a process allows *RecRules* to model rules in terms of shared functionality. A rule for turning on a *Philips Hue* lamp, for example, is *functionally* similar to a rule for opening the *Hunter Douglas* blinds, because they share a common final goal, i.e., to light up a place. The semantically enriched rules are then combined with collaborative information in a graph-based setting. *RecRules* extracts different types of features based on the paths, i.e., acyclic sequences of relationships between items, modeled in the graph. Such features are finally used to train a learning to rank algorithm and compute *top-N* recommendations.

4.2.1 Background

Recommendation opportunities in EUD have not yet been consistently explored, and most contemporary EUD solutions still continue to offer limited types of suggestions, e.g., by promoting the most popular rules, only. From the early works, EUD systems like EAGER [39] and Dynamic Macro [93] were using simple proactive suggestions to help end users define their programs. Moreover, in the last decade, recommendation technologies have been studied in the field of software engineering, mainly, from systems for feature recommendations [80, 65] to tools for source code suggestions [98]. The goal of these works, however, is to assist developers, instead of end users. In the same field, Ye and Fisher proposed *CodeBroker* [149], a development environment that autonomously locates and delivers task-relevant and personalized components into the current software development environment. Malheiros et al. [91], instead, developed a recommender system to help novice developers solve change requests in their source code. Other previous work targeting developers aim at suggesting APIs to facilitate expert-users in performing different tasks. Duala-Ekoko and Robillard [49] proposed an approach that leverages the structural relationships between API elements to make the related methods more discoverable. Nguyen et al. [106] presented a novel API recommendation approach, based on statistical learning, that taps into the predictive power of repetitive code changes to provide relevant API recommendations. D'Souza et al. [40] developed *PyReco*, an intelligent code completion system for Python that uses the mined API usages from open source repositories to order the results according to relevance rather than the conventional alphabetic order. Besides developers, only a

few recent works takes into account end users, e.g., by suggesting relevant smart "things" based on user preferences and interests, to optimize the time and cost of using IoT in a particular situation [147]. Instead of suggesting smart "things," *RecRules* suggests IF-THEN rules that relate pairs of connected entities: to the best of our knowledge, *RecRules* is one of the first examples of a recommendation algorithm that aims at helping users define IF-THEN rules for personalizing their smart devices and online services.

The algorithm is designed as a *top-N* semantic recommendation system. Referring to the *top-N* recommendation problem, several works have been proposed in the last few years. One of the most popular algorithm in this field is SLIM [107], an algorithm that uses a sparse linear method for learning a sparse aggregation coefficient matrix to be used for computing *top-N* recommendations. SLIM has been extended to incorporate both users and side information about items [108]. More recently, Wu et al. [145] proposed the Collaborative Denoising Auto-Encoder (CDAE) algorithm, a novel method for top-N recommendation that utilizes the idea of Denoising Auto-Encoders. Other works represented the *top-N* recommendation task as a ranking problem using *learning to rank*. Rendle et al. [118] proposed a Bayesian Personalized Ranking (BPR) criterion for optimizing a ranking loss. Also BPR has been extended in other works, e.g., to compute useful recommendations in cold start scenarios (BPR-MF [56]). Shi et al. [126] developed CLiMF, a novel collaborative filtering approach in which the model parameters are learned by directly maximizing the Mean Reciprocal Rank (MRR), which is a well-known information retrieval metric for measuring the performance of *top-N* recommendations. The same authors developed TFMAP [126], a model that directly maximizes Mean Average Precision with the aim of creating an optimally ranked list of items for individual users under a given context. TFMAP uses tensor factorization to model implicit feedback data (e.g., purchases, clicks) with contextual information. In our work, we compared *RecRules* with several state-of-the-art *top-N* recommendation algorithms, ranging from BPR-MF to Least Square SLIM [54].

Regarding semantic recommender systems, instead, several works have been proposed in the literature [124, 23, 2, 101] with the aim of improving recommendation performances, and to overcome some drawbacks of collaborative methods such as cold start and data sparsity. Furthermore, in the last 10 years, the advent of the Linked Open Data (LOD) [11] initiative opened the way for a new class of ontological recommender systems based on data freely available on the Web. One of the first approaches that exploits LOD to build a recommender system was the work of Heitmann and Hayes [66]. Here, the authors demonstrated that the usage of Linked Data mitigates the new-user, new-item, and sparsity problems of collaborative recommender systems. Fernández-Tobias et al. [51] showed a knowledge-based

framework for cross-domain recommendations leveraging DBpedia[6]. Khrouf and Troncy [75], instead, presented a novel hybrid approach built on top of Semantic Web for event recommendations. Their system was enhanced by the integration of a user diversity model designed to detect user propensity towards specific topics. Contextually to the progression of LOD-based recommender systems, recommendation methods based on generic heterogeneous networks have recently emerged. Knowledge graph embedding approaches, in particular, have proven to be effective to improve recommendations: they connect various types of information related to items (e.g., genre, director, actor of a movie) in a unified global space, which helps to develop insights on recommendation problems that are difficult to uncover with user-item interaction data only [132]. State of-the-art methods in this context mainly extend the latent factor model by considering similarities between items derived from paths in a knowledge graph. Yu et al. [150], for instance, presented a network-based entity recommendation method that uses path-based latent features to represent the connectivity between users and items along different types of paths. A global and local matrix factorization model, in particular, are learnt by using the BPR [118] approach. Lao and Cohen [82] improved the classic Random Walk with Restart approach by proposing a proximity measure defined as a weighted combination of path types, called *path experts*, obtained by fitting a logistic regression model. Ostuni et al. proposed SPRank [111], a hybrid recommender system to compute *top-N* recommendations from implicit feedback using linked data sources. SPRank directly formulates the problem of computing *top-N* recommendations in the standard learning to rank setting adopted in Web search [90] by replacing queries with users and document with items, and using user's ratings as relevance scores. It exploits an RDF graph, and it computes recommendations by training a learning to rank algorithm through path-based features. SPRank has also been extended to work with explicit feedback [109]. Similarly to SPRank, but with a more specific application area, Oramas et al. [110] described how to create and exploit a knowledge graph to supply a hybrid recommendation engine with information that builds on top of a collection of documents describing musical and sound items. They link the items to be recommended to external graphs such as WordNet and DBpedia through tags and textual descriptions, thus semantically enriching the initial data, and they add to the knowledge graph collaborative filtering information by connecting users to musical and sound items on the basis of their ratings. Then, they use the resulting knowledge graph in two versions of the algorithm, by formulating different explicit graph feature mappings based on entities and paths, respectively. Our approach is similar to previous works based on heterogeneous networks, in the way that we combine semantic and collaborative information to build a hybrid knowledge graph, and we use features based on the

[6] http://dbpedia.org, last visited on November 27, 2019

paths modeled in the graph for mining the user-item interactions. However, we exploit a semantic reasoning process on OWL ontologies to enrich IF-THEN rules with side information. Such a process allows the algorithm to uncover connections between rules in terms of shared functionality, and to capture the meaning of different types of path-based features. To implement the algorithm, in particular, we used the Linked Open Data Recommender Systems Library (lodreclib[7]) provided by the authors of SPRank [109].

4.2.2 Recommending By Functionality

RecRules aims to suggest IF-THEN rules on the basis of the final behaviors users would like to define, e.g., increasing the temperature in a room. In this way, *RecRules* can be exploited to compute recommendations for yet unknown or rarely used devices or services, thus helping users to discover new functionality, starting from their actual needs. The idea, in particular, is to abstract details such as involved technologies, brands or manufactures while maintaining the same low-level of abstraction of contemporary trigger-action programming platforms.

In these platforms, IF-THEN rules that differ from each other for some minor details, only, e.g., the manufacturer of a device, are considered different even if they are conceptually similar. A user that has already defined a rule for turning on her *Philips Hue* lamp in the kitchen, however, could be also interested in turning on other types of lamps, e.g., her *LIFX* lamp in the bedroom, or in opening the living room blinds. Specifically, she could be interested in the following rules:

R1 if the kitchen *Nest Cam* recognizes me, then turn on the kitchen *Philips Hue*;

R2 if the living room *Homeboy Cam* detects a movement, then open the *Hunter Douglas* blinds;

R3 if I open the *SmartThings* bedroom door, then turn on the bedroom *LIFX* lamp.

Even if they share a common goal, i.e., to light up a room, **R1**, **R2**, and **R3** are considered different in contemporary trigger-action programming platforms because they refer to different manufacturers. With such a technological-centric approach, the users experience with trigger-action programming platforms is very limited, and the actual end-user needs' are not taken into account. As suggested by Ur et al. [137], the continuous growth of trigger-action programming in the real world, and its application to a range of online services and physical devices, suggests the need to provide users with more support for discovering *functionality*, i.e., the behaviors that rules aim to define.

[7] https://github.com/sisinflab/lodreclib, last visited on November 27, 2019

To capture the similarities between **R1**, **R2**, and **R3**, thus characterizing them on the basis of their final purpose, we exploited a semantic reasoning process that enrich IF-THEN rules with semantic information. Different to previous works on semantic recommender systems, that simply associate items with relevant entities defined in online semantic datasets [58], the usage of reasoning capabilities allows *RecRules* to capture detailed information and to discover hidden relationships between IF-THEN rules, e.g., in terms of shared functionality. The reasoning process uses *EUPont*, with which the the similarities between **R1**, **R2**, and **R3** can be retrieved by semantically reasoning on the ML and HL classes of the involved triggers and actions. Both the actions *"turn on the Philips Hue lamp"* (**R1**) and *"turn on the LIFX lamp"* (**R3**), for example, are instances of the `Turn Lights On` action class, which is a subclass of the `Illuminate` action class. Furthermore, the action *"open the Hunter Douglas blinds"* (**R2**) is an instance of the `Open Blinds` class, which is a sibling class with respect to the `Turn Lights On` one, since the two ML classes share a HL class, i.e., `Illuminate`. The semantic reasoning process is detailed in Section 4.2.4.

4.2.3 Knowledge Graph Model & Problem Formulation

Linked Open Data (LOD) datasets can be viewed as vast decentralized knowledge graphs, where nodes correspond to RDF[8] entities, and labeled edges are properties connecting them. Furthermore, the semantic graph can be enriched with collaborative filtering information by embedding the collaborative filtering problem in the semantic graph as well, where users and items to be recommended are the nodes, and users' feedback, either implicit or explicit, are the links [109]. By reasoning on OWL ontologies, the semantic graph can be further enriched with the additional properties and features given by OWL classes and sub-classes (Figure 4.7).

The resulting multi-relational graph can be modeled as a directed graph $G = \{V, \Sigma\}$ where V denotes the set of vertices and Σ indicates the set of properties that connect them. In our model, the set of vertices is defined as $V = U \cup R \cup I \cup C$, where $u \in U$ are *users*, $r \in R$ are *rules*, $i \in I$ are RDF *individuals*, and $c \in C$ are OWL *classes*.

Semantic properties may have different nature, so each property is labeled with one label from the set $L = \{F, E, T\}$. The set of properties $\Sigma \subseteq V \times V \times L$ is the union of such different types of relationships, i.e., $\Sigma = \Sigma_F \cup \Sigma_E \cup \Sigma_T$:

- $\Sigma_F \subseteq U \times R \times \{F\}$ represents the set of *feedback relationships* (label F), which connects users to rules;

[8] https://www.w3.org/RDF/, last visited on November 13, 2019

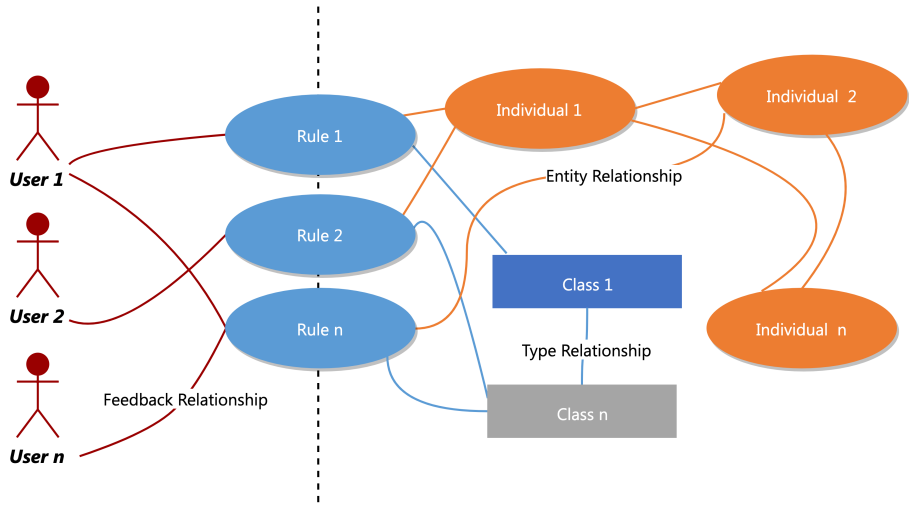


Figure 4.7: The knowledge graph built in *RecRules*. Users are connected to rules by means of feedback relationships. Rules are in turn connected to RDF individuals through individual relationships, and to OWL classes through class relationships

- $\Sigma_E \subseteq I \times R \times \{E\} \cup I \times I \times \{E\}$ represents the set of *entity relationships* (label E), which connects rules with RDF individuals, or RDF individuals among themselves;
- $\Sigma_T \subseteq C \times R \times \{T\} \cup C \times C \times \{T\}$ represents the set of *type relationships* (label T), which connects rules with OWL classes, or OWL classes among themselves.

Given the described model, the recommendation problem is formulated as follows. Considering the set of all users U, and the set of all rules R, the *user profile* of each user $u \in U$ is defined as $H_u = \{r \in R \mid \hat{r}_{ur} \in \hat{R}\}$, where:

- in case of explicit feedback, $\hat{R}$ is the user-item matrix, with each $\hat{r}_{ur} \in \mathbb{R}$ representing the rating of the user u on the rule r;
- in case of implicit feedback, $\hat{R}$ is the implicit feedback matrix, with each $\hat{r}_{ur} \in \mathbb{R}$ representing the implicit relevance of the rule r for the user u.

Regardless of the feedback type, the matching between user interests and rule content for each user-rule pair $(u, r) \in U \times R$ is mapped in the feature vector $\vec{x_{ur}} \in \mathbb{R}^D$, where D is the dimension of the feature space: each component $x_{ur} \in \mathbb{R}$ represents the connection of the rule r to the user u according to a specific feature k. The definition of the features is detailed in Section 4.2.4. In particular, we rely on different types of path-based features, able to characterize the interaction between users and rules with respect to collaborative information, technology-based

similarities, and functionality-based similarities. Eventually, the training set TS and the recommendation set RS are defined as:

$$TS = \bigcup_u \{< u, r, \vec{x_{ur}}, \hat{r}_{ur} > \mid r \in H_u\} \tag{4.14}$$

$$RS = \bigcup_u \{< u, r, \vec{x_{ur}}, \hat{r}^*_{ur} > \mid r \in (R \backslash H_u)\} \tag{4.15}$$

The final goal of the recommender system is to learn a scoring function from the training data TS able to predict $\hat{r}^*_{ur}$, in order to replicate for each user their perfect ranking.

4.2.4 The *RecRules* Algorithm

Figure 4.8 shows the architecture of *RecRules*. The algorithm has been implemented in Java by using the Linked Open Data Recommender Systems Library [109] (lodreclib[9]), and it is composed of two main phases, i.e., *Knowledge Graph Construction* and *Learning*, joined by a contextual filter (*User Context Filter*). The implementation we adopted for our experiments is available at https://git.elite.polito.it/public-projects/recrules.

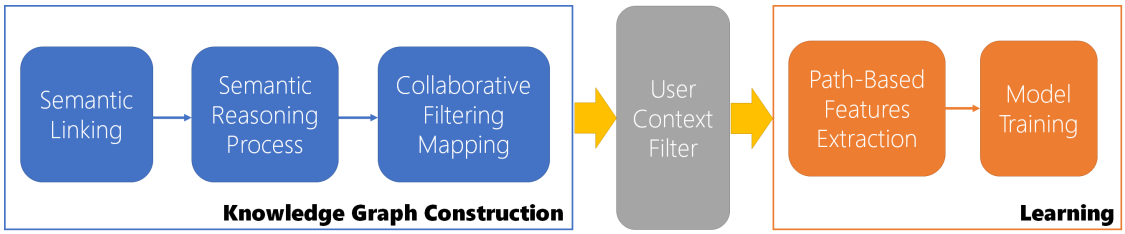


Figure 4.8: A schematic representation of the *RecRules* algorithm

Knowledge Graph Construction

Starting from a set of recommendable IF-THEN rules and the users' history, i.e., which rules have been already defined or reused, *RecRules* first build a knowledge graph that combines RDF individuals, OWL classes, and collaborative information. Individuals, in particular, represent the *technology information* that characterize the involved rules, i.e., low-level rules, triggers, actions, and involved connected entities. Classes, instead, represent the *functionality information* that are used to characterize the rules in terms of their final goal, independently of the involved technological details.

[9] https://github.com/sisinflab/lodreclib, last visited on November 13, 2019

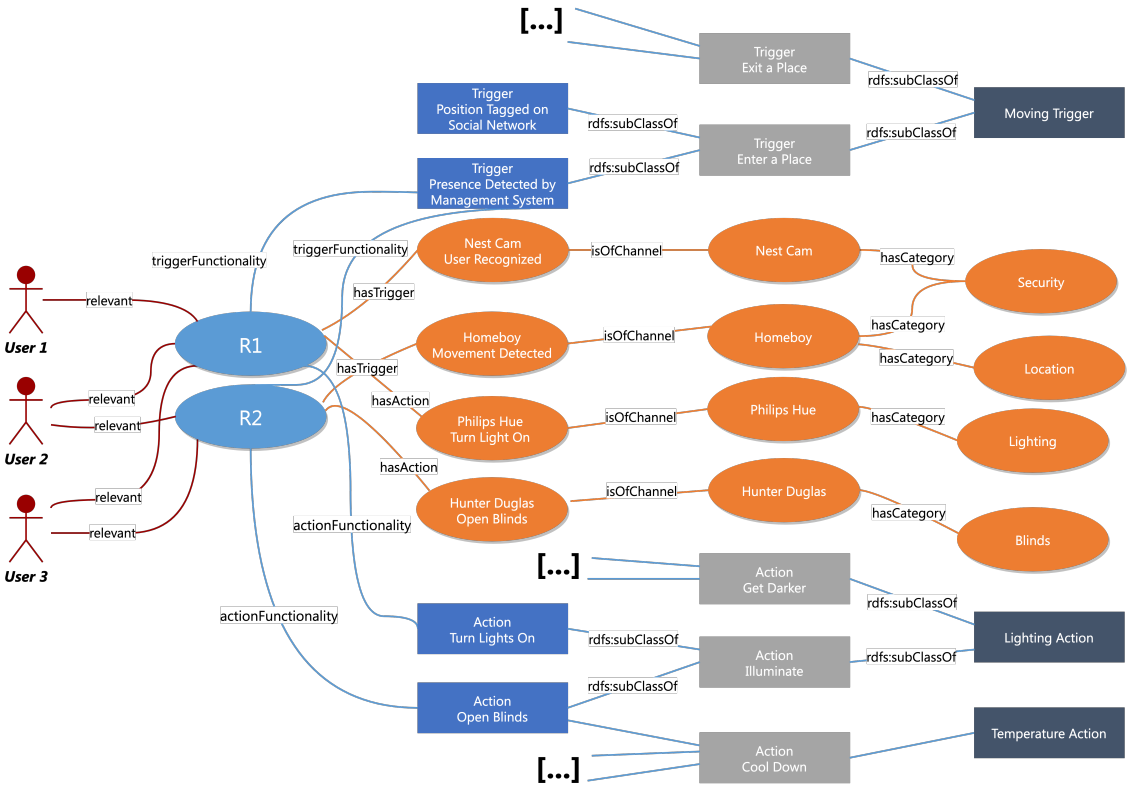


Figure 4.9: The knowledge graph built by *RecRules* by considering the rules R1 and R2. Rules are linked with individuals (ovals), i.e., the technology information, classes (rectangles), i.e., functionality information, and users, i.e., collaborative information.

Figure 4.9 shows an example of a simple knowledge graph built by considering R1 and R2, already reported in Section 4.2.2. Both rules have been extracted from the dataset exploited in the evaluation and rephrased for the sake of readability:

R1 if the kitchen *Nest Cam* recognizes me, then turn on the kitchen *Philips Hue*;

R2 if the living room *Homeboy Cam* detects a movement, then open the *Hunter Douglas* blinds.

Here, we describe the creation of the graph reported in the figure by highlighting its different parts, and we show how the graph is used in the recommendation process. In the example, we deliberately chose to represent a simple scenario with 2 users and 2 rules, only, to allow a better understanding of the underlying process. In a normal recommendation process, the algorithm models much more complex scenarios, where multiple users and rules are connected through different complex paths.

Semantic Linking & Graph Instantiation. To build a semantic graph such as the one reported in Figure 4.9, IF-THEN rules need to be linked to ontological sources. Different techniques and available ontological sources can be used for this purpose: the identification of entities in text-based resources is a well-known task in the Natural Language Processing community and it has recently gained momentum thanks to the availability of knowledge bases publicly available on the Web [125]. In our work, we link IF-THEN rules with the *EUPont* ontology. Since the dataset exploited in the evaluation (Section 4.2.5) is composed of trigger-action rules extracted from IFTTT, we exploit the instantiation of *EUPont* for IFTTT, that offers a hierarchical functionality representation of more than 500 triggers and actions supported by the popular platform. For linking rules to *EUPont*, in particular, we use the translation procedure described in Section 3.1.3, with which triggers and actions are linked to the corresponding ontological entities on the basis of their description and the devices involved and web applications. After the linking process, *RecRules* instantiates a semantic graph by adding the IF-THEN rules to be recommended, and by connecting them to the *technology information* that can already be extracted from contemporary trigger-action programming platforms. Such information are represented as RDF individuals, while their connections as entity relationships. In the graph of Figure 4.9 individuals are represented as ovals, and their connections as the `hasAction`, `hasTrigger`, `isOfChannel`, and `hasCategory` entity relationships. As zoomed in Figure 4.10, the rule *"if the kitchen's Nest Cam recognizes me, then turn on the kitchen Philips Hue"* (**R1**) is linked with the `Nest Cam User Recognized` trigger and with the `Philips Hue Turn Light On` action, for example.

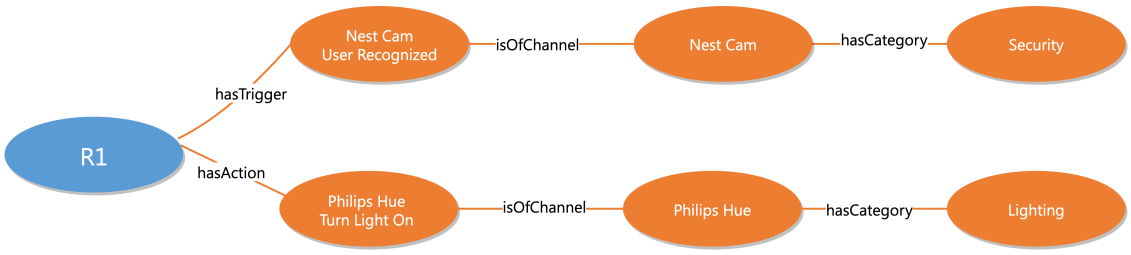


Figure 4.10: The figure is a zoom of Figure 4.9 that exemplifies the Semantic Linking & Graph Instantiation phase of *RecRules*. The rule **R1**, in particular, is linked to its trigger and its action by means of `hasTrigger` and `hasAction` entity relationships, respectively. The trigger and the action are in turn linked to other *technology information* through `isOfChannel` and `hasCategory` entity relationships, respectively.

Semantic Reasoning Process. To enrich IF-THEN rules with semantic information, and to characterize them in terms of shared functionality, we use a semantic reasoning process. The process analyzes triggers and actions of each rule, and recursively extracts their OWL ML and HL classes that represent the hierarchical characterization of triggers and actions offered by *EUPont* (*functionality information*, represented as rectangles in Figure 4.9). Such information are then linked in the form of OWL classes to the involved IF-THEN rules by means of the `triggerFunctionality`, `actionFunctionality`, and `subclassOf` type relationships. To implement the semantic reasoning process, *RecRules* uses OWL API [68], a high level Application Programming Interface (API) for working with OWL ontologies in Java, and the HermiT reasoner [61]. A semantic reasoner is a piece of software able to infer logical consequences from a set of asserted facts or axioms. HermiT, in particular, supports several specialized reasoning services such as class and property classification, as well as a range of features outside the OWL standard such as DL-safe rules and description graphs. We employ it to discover all the *EUPont* classes that can be used to characterize a trigger or an action, including the cases in which this information is not explicitly available, e.g., when a trigger or an action can be classified under different branches of the *EUPont* hierarchy. As zoomed in Figure 4.11, the rule *"if the living room Homeboy Cam detects a movement, then open the Hunter Douglas blinds"* (**R2**), for example, is connected both to the `Lighting Action` and the `Temperature Action` hierarchy branches.

Collaborative Filtering Mapping. The last part of Knowledge Graph Construction phase is the Collaborative Filtering Mapping, with which *RecRules* adds *collaborative information* to the semantic graph, i.e., how the rules are used and supported by the community of end users. In this way, the algorithm is able to discriminate between relevant rules, i.e., rules that are appreciated by the users, and not relevant rules, i.e., rules that are not appreciated and therefore should not be recommended. To add such feedback relationships, the algorithm defines a labeling function

$$\gamma : \mathbb{R} \longrightarrow \{\text{relevant, not relevant}\} \tag{4.16}$$

Different strategies can be used to define the labeling function, ranging from implicit feedback metrics (e.g., how many times a rule has been reused by other users), to explicit feedback metrics (e.g., rule ratings). In our evaluation of the approach (Section 4.2.5) we exploit a Graded Implicit Feedback [84] strategy by normalizing between 1 and 5 the number of times a rule has been reused by others.

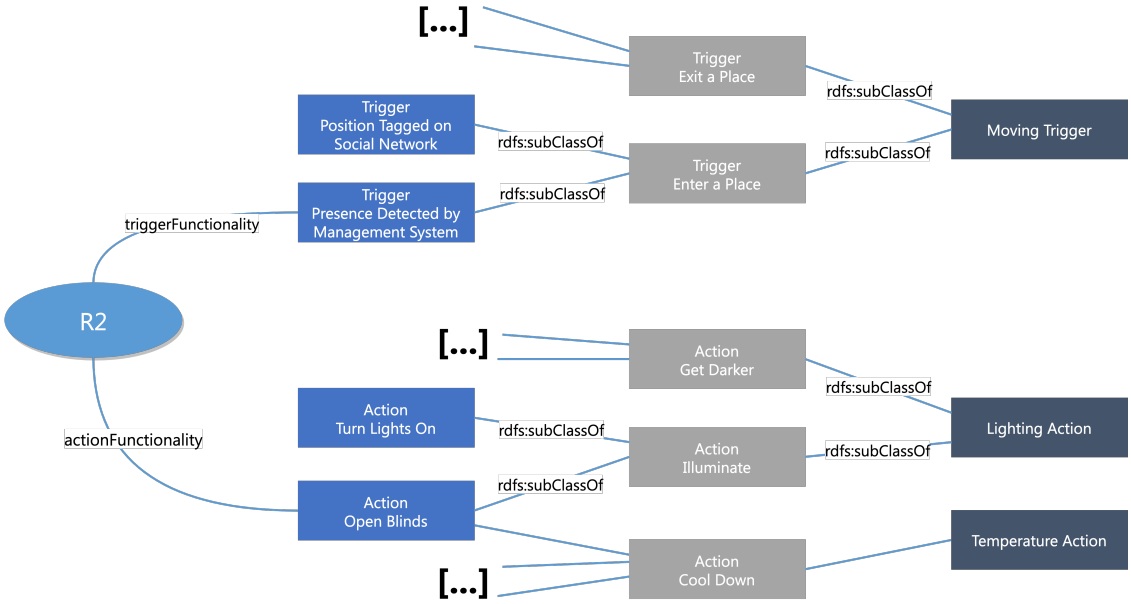


Figure 4.11: The figure is a zoom of Figure 4.9 that exemplifies the Semantic Reasoning Process phase of *RecRules*. The rule **R2**, in particular, is linked to the `Lighting Action` and the `Temperature Action` hierarchy branches of *EUPont* by means of `triggerFunctionality`, `actionFunctionality`, and `subclassOf` type relationships.

Learning

The generated semantic graph is then used in the Learning phase to extract different types of path-based features and to train a *learning to rank* model. Depending on the user for whom the recommendations need to be computed, the graph can be dynamically filtered through the *User Context Filter*. After the filtering process, the graph is restricted to IF-THEN rules that can be actually recommended to that user, i.e., the rules involving devices or web applications that the user is authorized to control.

Path-Based Features Extraction. To recommend IF-THEN rules, *RecRules* exploit the underlying connections modeled in the semantic graph. From the graph, in particular, the algorithm extracts path-based features able to characterize the interaction between users and rules. Given the sets of users U and rules R, a path $p_{u,r}$ is an acyclic sequence of adjacent relationships of length greater or equal than 2 that links a user $u \in U$ to a rule $r \in R$ in the semantic graph. The first step of the path $p_{u,r}$ links the user u to a rule belonging to her user profile H_u, while the rest of the path reaches the terminal rule r that does not necessarily belong to the user profile. Thanks to the information coded in the semantic graph, *RecRules* is

able to distinguish paths on the basis of their meaning:

Collaborative Paths. A collaborative path involves feedback relationships, only, i.e., it connects a user to a rule by means of other users. Considering the property labels L, collaborative paths have the form $(F, F, F, ..., F)$. Figure 4.12 shows a collaborative path extracted from the semantic graph of Figure 4.9. Here, u_1 is linked to r_2 by means of the path $p1_{u_1,r_2} = \{\texttt{relevant}, \texttt{relevant}^{-1}, \texttt{relevant}\}$. Besides this simple example, the algorithm is able to deal with more complex collaborative paths, i.e., that involve multiple rules connected through feedback relationships.

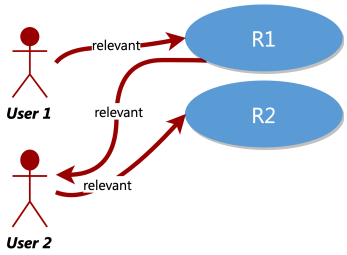


Figure 4.12: Connection between User 1 and Rule 2 through a collaborative path.

Technology Paths. A technology path is a path that, excluding the first relationship, involves entity relationships, only, i.e., it connects a user to a rule by means of technological information. Technology paths have the form $(F, E, E, ..., E)$. Figure 4.13 shows a technology path extracted from the semantic graph of Figure 4.9. Here, u_1 is linked to r_2 by means of the path $p2_{u_1,r_2} = \{\texttt{relevant}, \texttt{hasTrigger}, \texttt{isOfChannel}, \texttt{hasCategory}, \texttt{hasCategory}^{-1}, \texttt{isOfChannel}^{-1}, \texttt{hasTrigger}^{-1}\}$. Besides this simple example, the algorithm is able to deal with more complex technology paths, i.e., that involve multiple rules connected through entity relationships.

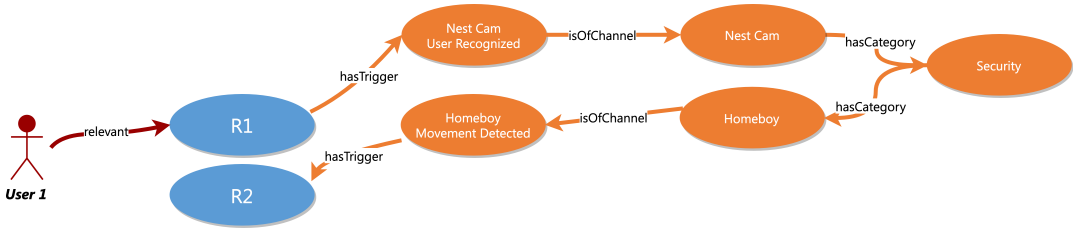


Figure 4.13: Connection between R1 and R2 through a technology path.

Functionality Paths. A functionality path is a path that, excluding the first relationship, involves type relationships, only, i.e., it connects a user to a rule by means of functionality information. Functionality paths have the

form $(F, T, T...T)$. Figure 4.14 shows a functionality path extracted from the semantic graph of Figure 4.9. Here, u_1 is linked to r_2 by means of the path $p3_{u_1,r_2}$ = {`relevant`, `actionFunctionality`, `subClassOf`, `subClassOf`$^{-1}$, `actionFunctionality`$^{-1}$}. Also in this case, besides this simple example, the algorithm is able to deal with more complex technology paths, i.e., that involve multiple rules connected through type relationships.

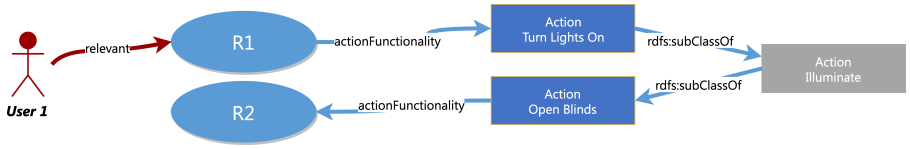


Figure 4.14: Connection between R1 and R2 through a functionality path.

We define the signature k of a path $p_{u,r}$ as the sequence of the actual properties traversed by the path. The signature $k = 1$ of the path $p1_{u_1,r_2}$ is, for example, {*relevant*, `relevant`$^{-1}$, *relevant*}. Each distinct signature k maps to a distinct dimension $x_{ur}(k)$ of the feature vector $\vec{x_{ur}}$. Considering a path with signature k, in particular, the feature component $x_{ur}(k)$ is computed by counting $\#p_{u,r}(k)$, i.e., how many *instances* of paths having signature k exist between a user u and a rule r. For the graph of Figure 4.9, for example, $\#p_{u_1,r_2}(1) = 2$, since there are 2 instances of the path {*relevant*, `relevant`$^{-1}$, *relevant*} that connect u_1 to r_2. To compare features between different users, we follow the strategy of Di Noia et al. [109] by introducing a user-based normalization. At the end, the components of the feature vector are computed with the following formula:

$$x_{ur}(k) = \frac{\#p_{u,r}(k) - \min_{w \in R}\left(\#p_{u,w}(k)\right)}{\max_{w \in R}\left(\#p_{u,w}(k)\right) - \min_{w \in R}\left(\#p_{u,w}(k)\right)} \tag{4.17}$$

Equation (4.17) represents the importance of the path of type k between user u and rule r.

Model Training. To finally compute *top-N* recommendations, a ranking model from training data is built using a learning to rank technique. The goal of the Model Training phase is to learn a scoring function in such a way the model can sort new items according to their relevance. Different techniques can be used for this purpose. They can be classified into three main categories: pointwise, pairwise, and listwise. In our evaluation of *RecRules* (Section 4.2.5) we explore three popular learning to rank algorithms that can be considered as the baseline in the corresponding learning to rank category, namely Random Forest [15], RankBoost [53], and LambdaMart [144].

Pointwise Learning to Rank. Pointwise approaches look at a single instance at a time, and transform the ranking problem into a regression or a classification

one. In particular, they take a single instance at a time, and train a classifier (or a regressor) to predict its relevance. Each instance is therefore independent from each other, and the final ranking is achieved by simply sorting the result list looking at the predicted scores. For the pointwise category, *RecRules* uses Random Forest, an algorithm that constructs a multitude of decision trees at training time and outputs the class label for classification (or the mean prediction for regression) of the individual trees.

Pairwise Learning to Rank. Pairwise approaches look at a pair of instances at a time, and try to find out their optimal ordering. The goal for a pairwise ranker is to minimize the number of inversions in ranking, i.e., cases where the pair of results are in the wrong order relative to the ground truth. For the pairwise category, *RecRules* uses RankBoost, an algorithm that builds a linear combination of weak rankers, and optimizes a loss function based on the exponential difference between the relevance of pairs of items.

Listwise Learning to Rank. Listwise approaches directly look at the entire list of instances. In particular, they try to come up with the optimal ordering by minimizing a loss function defined over the ranked list of instances. For the listwise category, *RecRules* uses LambdaMart (LMART), an algorithm that exploits the normalized Discounted Cumulative Gain (nDCG) metric for fitting the parameters of the regression trees

4.2.5 Algorithm Evaluation

We evaluated *RecRules* through different experiments. First, we assessed the effectiveness of *recommending by functionality* by exploring the accuracy of 3 learning to rank techniques trained with different types of path-based features. Finally, we compared *RecRules* with state-of-the-art collaborative filtering, ranking-oriented, and semantic-aware recommendation algorithms. Two research questions, in particular, guided the study:

RQ7) To what extent the different types of path-based features influence the recommendation accuracy?

RQ8) Does *RecRules* outperform state-of-the-art recommendation systems in suggesting IF-THEN rules?

Dataset

The evaluations were based on the IFTTT dataset of Ur et al. [137]. To the best of our knoweldge, this is the only publicly available dataset of IF-THEN rules defined and shared by different users. As already reported in Section 3.1.3, the

dataset contains 295,156 rules created and shared by 129,206 different authors, and has a high degree of sparsity (97.51%, Table 4.3).

Table 4.3: IFTTT dataset statistics.

Metric	Value
Users (#)	129,206
Items (#)	295,156
Sparsity (%)	97.51

Table 4.4: Example of a rule stored in the IFTTT dataset exploited for the evaluation.

Field	Value
Id	100301
Description	Save Soundcloud likes to Google Drive.
Author	gigaphon
Trigger Channel	SoundCloud
Trigger	New public like
Trigger Description	This Trigger fires every time you like a public track.
Action Channel	Google Drive
Action	Upload file from URL
Action Description	This Action will download a file at a given URL and add it to Google Drive at the path you specify. NOTE: 30 MB file size limit.
Shares	2.2k

Table 4.4 shows a rule extracted from the dataset. Beside the information about the trigger (Trigger Channel, i.e., involved connected entity, Trigger, and Trigger Description) and the action (Action Channel, i.e., involved connected entity, Action, and Action Description), each rule includes data about the author who created it (Author), a description of the entire rule (Description), and the information about how many times the rule has been reused by other users (Shares). According to the dataset, some rules were reused more than 400,000 times, while others were reused only by one user (M=837.79, SD=9452.77, $range$=1 : 476355). We used this information to calculate the labeling function γ (Eq. 4.16), in order to define relevant and not relevant rules. As suggested in [84], instead of randomly choosing negative data points, we computed a Graded Implicit Feedback (GIF) by normalizing the number of reuse between 1 and 5. Then, we defined γ as

$$\gamma = \begin{cases} \text{relevant}, & \text{if GIF} > 3 \\ \text{not relevant}, & \text{otherwise} \end{cases} \tag{4.18}$$

The threshold was empirically set to 3 to obtain two homogeneous groups. Our idea was to promote rules that were already reused by a consistent number of users. Table 4.5 reports the distribution stemming from the labeling function γ: at the end, 45.88% of the rules were considered as "not relevant", while the remaining 54.12% were considered as "relevant." Finally, since the dataset does not contain any information about which smart devices or online services can be actually controlled by each user, we set up the *User Context Filter* by supposing that each user is authorized to control *any* connected entity. For all the experiments, we follow a k-fold cross-validation approach by randomly splitting up the dataset into 10 groups. Similarly, the tuning of the parameters in the learning to rank algorithms was performed through cross validation on validation data obtained by selecting 15% of items for each user from the original dataset.

Table 4.5: Distribution of the computed Graded Implicit Feedback (GIF), obtained by normalizing between 1 and 5 the number of shares.

GIF	Label	Coverage
1	not relevant	32.46%
2	not relevant	13.42%
3	relevant	5.38%
4	relevant	14.3%
5	relevant	34.44%

Procedure

In all the evaluations, we used the "all unrated items" methodology [129], that consists in computing a *top-N* recommendation list for each user and by predicting a score for every item not rated by that particular user, including those that are not in the test set. The main assumption in this methodology is that all the unrated items are considered to be irrelevant for the user, with the effect of underestimating the recommendation quality. However, the user experience in *top-N* recommendation applications depends on the ranking of all items. This implies that, in this case, the "all unrated items" methodology is more suitable than the "rated test-items," where only the test data are considered for generating the *top-N* recommendations [129].

Metrics

We used different metrics for investigating the results. All of them were computed for each single user and then averaged. In particular, we measured the recommendation accuracy with three standard performance metrics, i.e., precision, recall, and normalized discounted cumulative gain.

For the definition of the metrics, we considered:

- R, i.e., the set of all the trigger-action rules;
- S, i.e., the set of *top-N* recommendations;
- $m^{+,N}$, i.e., the number of recommendations in S that are relevant;
- m^{+}, i.e., the number of all relevant rules.

Precision represents the fraction of the *top-N* recommended rules that are relevant among all the recommended items S. It is computed with the formula

$$prec@N = \frac{m^{+,N}}{N} \tag{4.19}$$

Recall represents the fraction of the *top-N* recommended rules that are relevant over the total amount of relevant rules. The recall is computed with the formula

$$rec@N = \frac{m^{+,N}}{m^{+}} \tag{4.20}$$

Different from precision and recall, which are binary metrics that consider the relevance of the items, only, **normalized Discounted Cumulative Gain (nDCG)** can handle graded values by taking into account both relevance and rank position. Given $\hat{r}_{ur_j}$, i.e., the GIF of u to the rule r_j in the j-th position of the recommended rules S, and $IDCG@N$, i.e., a normalization factor that represents the score obtained by an ideal or perfect *top-N* ranking, $nDCG@N$ is computed with the formula

$$nDCG@N = \frac{1}{IDCG@N} \sum_{j=1}^{N} \frac{2^{\hat{r}_{ur_j}} - 1}{\log_2(1+j)} \tag{4.21}$$

In addition to precision, recall, and nDCG, we used diversity, coverage, and serendipity metrics to investigate our results beyond accuracy. Standard accuracy metrics, in fact, cannot measure all the different aspects of a recommendation process, and the recommendations that are most accurate according to the standard metrics are

sometimes not the recommendations that are most useful to users [97].

Diversity measures how dissimilar recommended items are for a user. A popular metric for measuring diversity is the Intra-List Similarity (ILS) [156]. We used the Jaccard similarity coefficient J to calculate the similarity between 2 rules in terms of technologies, i.e., involved smart devices or online services. We firstly defined $\tau(r)$ as the set of connected entities involved in triggers and actions of a given rule r. Then, we define ILS as

$$ILS@N = \frac{1}{2} \sum_{r_i \in S} \sum_{r_j \in S} J(\tau(r_i), \tau(r_j)) \tag{4.22}$$

The Jaccard similarity coefficient measures similarity between finite sample sets, e.g., A and B. The coefficient is defined as the size of the intersection divided by the size of the union of the sample sets:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} \tag{4.23}$$

In our case, $A = \tau(r_i)$ and $B = \tau(r_j)$ represent the set of connected entities involved in triggers and actions of 2 different rules.

Coverage represents the percentage of things (items, users, or ratings) that the recommender system is able to recommend. Not being able to predict a particular set of users or items is usually caused by an insufficient number of ratings, and is generally known as the cold start problem. In our work, we used the item coverage metric, i.e., the number total number of recommended rules (n) over the total number of trigger-action rules

$$COV@N = \frac{n}{|R|} 100 \tag{4.24}$$

Serendipity is the measure of how surprising the successful or relevant recommendations are. We assessed serendipity through the unserendipity metric [155], by using the Jaccard similarity coefficient J to measure the technological similarity in terms of involved connected entities between rules in the user's profile (H_u) and new recommendations in S. Lower values of this metric indicate that recommendations deviate from a user's traditional behavior, and hence are more surprising. For a given user u, in particular, unserendipity is defined as

$$UNSER@N = \frac{1}{|H_u|} \sum_{r_i \in H_u} \sum_{r_j \in S} \frac{J(\tau(r_i), \tau(r_j))}{20} \tag{4.25}$$

4.2.6 Results

Approach Effectiveness

We analyzed the main characteristic of *RecRules*, i.e., recommending by functionality, by exploring to what extent the different types of path-based features influence the recommendation process, and, in particular, recommendation accuracy (**RQ7**).

For this purpose, we compared the accuracy of the 3 learning to rank techniques implemented in *RecRules*, i.e., Random Forest, RankBoost, and LambdaMart, in two different configurations, by changing the Learning phase of *RecRules*. In particular, *i)* we first trained the algorithms by exploiting collaborative and technology paths, only (*CT* configuration), and *ii)* we then trained the same algorithms by using all the collaborative, technology, and functionality paths (*CTF* configuration), with the aim of understanding whether the functionality information improved the recommendation process. Table 4.6 reports the results of such a comparison in terms of precision, recall, and nDCG. As shown in the table, the exploitation of functionality paths in the recommendation process resulted in an increase of the recommendation accuracy: except for *nDCG*@10 of LambdaMart, which values are almost equal, all the accuracy metrics are better in the *CTF* configuration than the *CT* configuration. This suggests that, by extracting similarities between rules in terms of shared functionality, *RecRules* is able to uncover useful hidden connections between rules that cannot be identified in contemporary trigger-action programming platforms. This introduces different advantages: the algorithm can overcome technological constraints and suggest IF-THEN rules even for rarely used devices and web applications, on the basis of the actual user's needs.

Another result that can be extracted from Table 4.6 is that Random Forest and LambdaMart (the pointwise and listwised approaches, respectively) provided similar results, and performed better than RankBoost (the pairwise approach) for all the metrics and configurations. Random Forest in the *CTF* configuration, in particular, was selected as the winning configuration since it provided the best results in general. The Random Forest algorithm, indeed, works especially well for high-variance, low-bias datasets (as the one exploited in this evaluation), and it offers a good handling of missing data.

To further investigate the potential of *recommending by functionality*, we analyzed the recommendations computed by *RecRules* in its 2 configuration, i.e., *CT* and *CTF*, with diversity (ILS), coverage (COV), and serendipity (UNSER) metrics. The goal was to go beyond accuracy, thus identifying other advantages of recommending by functionality. Table 4.7 reports the results for the ILS, COV, and UNSER metrics using Random Forest. While the 2 configurations resulted in recommendations with similar coverage and serendipity, recommendations computed

Table 4.6: Comparative results in terms of precision, recall, and nDCG of the three learning to rank algorithms implemented in *RecRules* in two different configurations, i.e., CT (collaborative and technology features) and CTF (collaborative, technology, and functionality features). Bold numbers are used to highlight the best results for to the same learning to rank algorithm, while gray cells indicate the best results across all the evaluated algorithms. Results are the average of 5 equal experiments.

Algorithm		prec@5	rec@5	nDCG@5	prec@10	rec@10	nDCG@10
Random Forest	***CT***	0.1077	0.2090	0.4920	0.0772	0.2901	0.5830
	CTF	**0.1211**	**0.2177**	**0.5054**	**0.0813**	**0.3019**	**0.6452**
Rank Boost	***CT***	0.0743	0.1309	0.4558	0.0570	0.1998	0.5515
	CTF	**0.0967**	**0.1894**	**0.4861**	**0.0660**	**0.2536**	**0.5753**
Lambda Mart	***CT***	0.0900	0.1918	0.4884	0.0633	0.2589	**0.5756**
	CTF	**0.1115**	**0.2123**	**0.4893**	**0.0858**	**0.2941**	0.5754

by *RecRules* in the *CTF* configuration were less similar in terms of involved connected entities with respect to the suggestions proposed by *RecRules* in the *CT* configuration ($ILS = 0.0439$ *vs.* $ILS = 0.1088$, respectively). This confirms that, by using functionality-based features, *RecRules* encourages the recommendation of IF-THEN rules for controlling new devices and online services, without affecting accuracy.

Table 4.7: Diversity (ILS metric), coverage (COV metric), and serendipity (UNSER metric) results for CT and CTF configurations using Random Forest. Bold numbers are used to highlight the best results. Results are the average of 5 equals experiments.

Configuration	ILS@10	COV@10	UNSER@10
CT	0.1088	**13.94%**	0.6774
CTF	**0.0439**	13.57%	**0.6243**

Comparison With Other Algorithms

Beside investigating the effectiveness of the functionality-based features, we compared *RecRules* in its best setting, i.e., Random Forest in the *CTF* configuration, with state-of-the-art collaborative filtering, ranking-oriented and semantic recommendation algorithms (**RQ8**), namely:

- **Item-KNN.** The Item-KNN algorithm used for the evaluation is a baseline

item-based K-Nearest Neighbours (KNN) algorithm [57]. KNN is a non-parametric, lazy learning method that uses a database in which the data points, i.e., the items, are separated into several clusters to make inference for new samples. Item-KNN does not make any assumptions on the underlying data distribution but it relies on item feature similarity.

- **User-KNN.** The User-KNN algorithm used for the evaluation is a baseline user-based K-Nearest Neighbours algorithm [57]. Differently from Item-KNN, User-KNN recommends items by analyzing similar users: it firstly finds the K-nearest neighbors to a specific user a, and it then predicts the rating that a will give to all items the k neighbors have consumed but a has not.

- **Soft Margin Ranking MF (SMR MF).** The SMR MF is a matrix factorization model for item ranking which uses ordinal regression score as loss function [142]. Matrix factorization methods represent the state-of-the-art for rating prediction tasks.

- **BPR-MF.** The BPR-MF algorithm [56] is a hybrid extension of the Bayesian personalized ranking (BPR) [118] that learns a linear mapping on the user/item features from the factorization matrix. This extension of BPR is able to compute useful recommendations in cold-start scenarios.

- **BPR-SLIM.** The BPR-SLIM algorithm is an extension of the SLIM algorithm [107] that uses the BPR criterion. SLIM, in particular, uses a Sparse Linear method for learning a sparse aggregation coefficient matrix.

- **WRMF.** The WRMF algorithm [69] is a weighted matrix factorization method that interprets the number of times an item is observed by a user as a measure for the user preference, and uses regularization to prevent overfitting.

- **Least Square SLIM (LS SLIM).** The LS SLIM algorithm is a variant of the SLIM algorithm in which the the model is learned using a coordinate descent algorithm with soft thresholding [54].

- **Item Attribute KNN (IA KNN).** The IA KNN algorithm used for the evaluation is an attribute-based K-Nearest Neighbours approach [57]. Besides items, users, and ratings, the algorithm can access other types of side information in the form of item attributes. In our evaluation, we provided the algorithm with the semantic information used in *RecRules*, i.e., OWL classes and super-classes of each trigger and action according to the EUPont ontology.

- **BPR-Linear.** BPR-Linear [56] is a hybrid matrix factorization method able to work with sparse datasets. As for the Item Attribute KNN algorithm, BPR-Linear is able to manage side information. Also in this case, we enriched the

algorithm with the semantic information provided by EUPont.

- **Entity Graph-Embedding (EGE).** The EGE algorithm is a hybrid *semantic* recommender system proposed by Oramas et al. [110]. The algorithm exploits a knowledge graph and two different embedding approaches to encode knowledge graph information into a linear feature representation. In the Entity Graph-Embedding, in particular, features are calculated by analyzing the neighborhood of each entity composing the knowledge graph.

To compute the recommendations with Item-KNN, User-KNN, Soft Margin Ranking MF, BPR-MF, BPR-SLIM, WRMF, Least Square SLIM, Item Attribute KNN, and BPR-Linear we used *MyMediaLite* [57], a publicly available software library for recommender systems. For implementing the Entity-Based Graph-Embedding algorithm, instead, we used the *lodreclib* library [109], by building the same knowledge graph built in *RecRules*, i.e., with the semantic information of *EUPont*.

Table 4.8: Comparison of *RecRules* with other state-of-the-arts algorithms in terms of precision, recall, and normalized discounted cumulative gain on top-N recommendations (with N=5 and 10). Results are the average of 5 equals experiments.

	prec@5	**rec@5**	**nDCG@5**	**prec@10**	**rec@10**	**nDCG@10**
RecRules	**0.1211**	**0.2177**	**0.5054**	**0.0813**	**0.3019**	**0.6452**
Item-KNN	0.0847	0.1807	0.1939	0.0514	0.2383	0.2095
User-KNN	0.0961	0.2103	0.2410	0.0520	0.2277	0.2419
SMR MF	0.0760	0.1716	0.1942	0.0452	0.2019	0.1905
BPR-MF	0.1085	0.1898	0.2082	0.0664	0.2148	0.2131
BPR-SLIM	0.1110	0.1976	0.2224	0.0616	0.2200	0.2216
WRMF	0.1155	0.2045	0.2228	0.0618	0.2217	0.2223
LS SLIM	0.1105	0.1970	0.2196	0.0604	0.2158	0.2229
IA KNN	0.0273	0.0845	0.2398	0.0207	0.1302	0.2357
BPR-Linear	0.0504	0.1708	0.2957	0.0356	0.2383	0.2890
EGE	0.0975	0.1918	0.4728	0.0656	0.2467	0.5625

Table 4.8 reports the results of the comparison evaluation in terms of accuracy metrics for all the algorithms. Looking at the results, our algorithm outperformed the algorithms that do not take into account semantic information in computing recommendations. In particular, *RecRules* performed better than baseline collaborative filtering methods (i.e., Item-KNN and User-KNN) and Matrix Factorization approaches (i.e., Soft Margin Ranking MF, BPR-MF, and WRMF) in terms of precision, recall, and nDCG. Furthermore, it also outperformed other hybrid learning to rank methods such as BPR-SLIM and Least Square SLIM. This further confirms

the usage of semantic information capturing functionality similarities improved the recommendation accuracy (**RQ7**).

The comparison of *RecRules* with Item Attribute KNN and BPR-Linear, i.e., the 2 algorithms that used semantic information as item attributes, highlights that the potential of our approach is not the usage of semantic information, only, but is its usage as a graph. Indeed, the usage of the underlying knowledge graph, along with the extraction of different path-based features, provided clear advantages in terms of precision, recall, and nDCG with respect to Item Attribute KNN and BPR-Linear.

The benefits of using a graph-based model, in particular, can be glimpsed by looking at the results of the Entity Graph-Embedding (EGE) approach, i.e., the other algorithm able to exploit the same knowledge graph built in *RecRules*. EGE, in particular, outperformed all the other evaluated state-of-the-art algorithms in terms of recommendation accuracy, and provided results in line with those obtained with *RecRules*, especially for what concern the nDCG metric. The high nDCG values of EGE and *RecRules*, in particular, suggest that the usage of an underlying knowledge graph positively influences the capability of the 2 algorithms in ranking IF-THEN rules by relevance. Moreover, precision, recall, and nDCG were slightly higher with *RecRules*: we can reasonably conclude that using the knowledge graph to capture connections between IF-THEN rules in terms of shared functionality, i.e., functionality paths, is a promising approach.

To further investigate the potential of *RecRules*, we repeated our analysis of diversity, coverage, and serendipity by looking at the recommendations of the previous state-of-the-art approaches. Table 4.9, in particular, reports the results of the ILS, COV, and UNSER metrics for all the evaluated algorithms.

Only the recommendations computed with EGE covered a higher number of rules with respect to *RecRules* ($COV = 18.72\%$ *vs.* $COV = 13.57\%$, respectively), but its recommendations were in general less surprising than the rules suggested by *RecRules* ($UNSER = 0.9812$ *vs.* $UNSER = 0.6243$, respectively). Looking at the table, also the recommendations computed by the other 2 algorithms that used semantic information, i.e., Item Attribute KNN and BPR-Linear, were in general less surprising than the rules suggested by *RecRules*: this may suggest that using different semantic path-based features, based on technology, collaborative, and functionality information, resulted in a higher serendipity. Furthermore, even if the *RecRules* suggestions were in general less surprising than the rules suggested by the other collaborative filtering and ranking-oriented methods, i.e., Item-KNN, User-KNN, Soft Margin Ranking MF, BPR-MF, BPR-SLIM, WRMF, and Least Square SLIM, recommendations computed by *RecRules* were less similar in terms of involved technologies, brands, and manufacturers with respect to the majority of the other evaluated algorithms: only for the Item-KNN and Soft Margin Ranking MF the ILS metric was slightly lower.

Table 4.9: Comparison of *RecRules* with other state-of-the-arts algorithms in terms of diversity (ILS metric), coverage (COV metric), and serendipity (UNSER metric) on top-N recommendations (with N=10). Results are the average of 5 equals experiments.

	ILS@10	**COV@10**	**UNSER@10**
RecRules	0.0439	13.57%	0.6243
Item-KNN	**0.0362**	4.53%	0.0167
User-KNN	0.0476	3.14%	**0.0129**
SMR MF	0.0419	3.64%	0.0208
BPR-MF	0.0571	3.66%	0.5083
BPR-SLIM	0.0685	5.18%	0.5360
WRMF	0.0857	4.28%	0.5253
LS SLIM	0.0476	3.46%	0.4851
IA KNN	0.1125	12.01%	1.1616
BPR-Linear	0.1328	11.90%	1.2695
EGE	0.0471	**18.72%**	0.9812

4.3 Discussion and Guidance for Future Research

With the representation models adopted by contemporary trigger-action programming platforms, where connected entities are modeled on the basis of the underlying brand or manufacturer, the number of supported triggers and actions is continuously growing, and users experience difficulties in finding the functionality they need to define their IF-THEN rules (*information overload* issue). In this chapter, we investigated two different approaches to support end users in discovering and managing rules and related functionality, i.e., *EUDoptimizer* and *RecRules*.

EUDoptmizer. We proposed *EUDoptimizer* to interactively assist end users in defining IF-THEN rules with an optimizer in the loop. The goal is to *dynamically* redesign layouts in a trigger-action programming interface in an *interactive* way, i.e., by considering the choices made by the user, so that that the needed functionality are immediately displayed on the top of the interface. To reach our goal, we adapted a state-of-the-art predictive model of user performance in menu search, and we defined a novel model to organize connected entities (i.e., various smart devices, online services, ...) on the basis of their final functionality. We used different optimization algorithms to explore the design space, and we integrated the optimization methods on top of IFTTT. The *EUDoptimizer* implementation, in particular, suggests that the approach is valuable. *Off-line* results obtained with 10,000 iterations (~15/20 minutes on a regular laptop) were promising, and showed that satisfactory solutions

can be obtained in a reasonable amount of time. This is confirmed by the results of the user study, where *EUDoptimizer* performed the optimizations in *real-time*, by interacting with the participants. By comparing *EUDoptimizer* with *IFTTT*, in particular, we found that IF-THEN rules were defined in less time with the optimized interface. Even for the most uncommon rules, for which the involved connected entities were not placed on the top of the layouts, *EUDoptimizer* partially reduced the time effort needed by participants to complete the tasks. Furthermore, qualitative data extracted from the user evaluation suggest that *EUDoptimizer* reduces the cognitive effort to define IF-THEN rules by redesigning the grid layouts of user interfaces for trigger-action programming with a focus on the final *functionality* of the supported technologies.

RecRules. As suggested by previous work [64], an alternative approach to functionality discovery is to directly suggests proper IF-THEN rules to be activated, on the basis of the user's needs. To this end, we presented *RecRules*, our hybrid and semantic recommendation algorithm for suggesting IF-THEN rules. Through a mixed content and collaborative approach, *RecRules* exploits different path-based features and learning to rank techniques, and it is able to interact with OWL ontologies to compute *top-N* recommendations. For the semantic part, we exploited the *EUPont* model. With such a representation, *RecRules* not only takes into account connected entities already used by users, but is able to recommend IF-THEN rules on the basis of their final functionality, i.e., the behavior that they aim to define. By exploiting a dataset of trigger-action rules created and shared by real users on IFTTT, we demonstrated the effectiveness of our approach by evaluating three different learning to rank algorithms, and by investigating to what extent the different path-based features affected the computed recommendations. Furthermore, we showed that *RecRules* outperforms state-of-the-art ranking-oriented and semantic recommendation algorithms. In terms of accuracy metrics, i.e., precision, recall, nDCG, *RecRules* outperformed all the other evaluated approaches, ranging from baseline collaborative filtering methods, e.g., Item-KNN and User-KNN, to other semantic-based approaches that used the same underlying semantic information, e.g., Entity Graph-Embedding. Furthermore, the usage of different semantic path-based features increased the coverage and the diversity of the computed recommendations, by allowing *RecRules* to remain competitive (and in some cases better) in terms of serendipity. Moreover, functionality paths increased the recommendation accuracy of the explored learning to rank approaches: this suggest that the main characteristic of *RecRules*, i.e., *recommending by functionality*, is effective in suggesting IF-THEN rules, and could help end users

discover new rules based on their final purpose, rather than the involved devices and online services. Our approach is therefore able to uncover useful hidden connections between rules that cannot be identified in contemporary trigger-action programming platforms and through conventional and established recommendation systems based on popularity or involved technologies. Such a feature can be glimpsed by qualitatively analyzing the recommendations computed by *RecRules*. For a specific user, for example, we found the following recommended rule: *"if my Nest detects a smoke alarm, then send me an Android SMS"*. By analyzing the training set for that user, we found that it contained *"if the Scout Alarm triggers, then send me a notification on my Google Glasses"*, that is conceptually equivalent to the recommended rule in terms of final functionality, i.e., "let me know if something is wrong in my home," but includes different technologies, i.e., *Scout Alarm* and *Google Glasses*.

Results presented in this chapter point to novel trigger-action programming platforms that proactively support users in discovering the functionality they need. On the one hand, integrating optimization methods in trigger-action programming interfaces could help end users better deal with trigger-action programming, and could open up new possibilities for users to program their devices and services. Besides reorganizing grid layouts to display connected entities, as in *EUDoptimizer*, several applications of optimization methods in trigger-action programming interfaces can be further explored in future works, e.g., to display relevant information, only, or to dynamically change the rule definition process.

On the other hand, the adoption of recommender systems in the EUD context could effectively help end users define their applications. Few EUD systems today take advantage of recommender technologies [64], and the most common trigger-action programming platforms still continue to offer limited types of suggestions. With *RecRules*, we hope to open the way to a new class of recommender systems in EUD based on the actual end-users' needs. While a conventional recommender system would probably suggest, regardless of the user goal, rules that involve the same connected entities, *RecRules* is able to establish, in high-level terms, what the user is trying to achieve with the rules she has already defined. Future works would need to assess the *RecRules* algorithm with further evaluations, e.g., by exploiting data coming from different platforms. For this purpose, a urgent challenge is to provide the scientific community with more datasets in the trigger-action programming context. Indeed, differently from other domains such as movies and songs, recommendations in the EUD are in their early stages, and, to the best of our knowledge, the IFTTT dataset of Ur et al. [137] is the only publicly available collection of IF-THEN rules. Besides testing *RecRules* offline, future works would need to perform more user-centered evaluations of the approach, e.g, by integrating it in a real platform to define trigger-action rules. Recommendations computed by

RecRules could be used in a "traditional" way, e.g., by displaying them in the home page of the interface, or they could be used to assist users in the rule definition process. An example of a possible integration of *RecRules* for supporting the definition of IF-THEN rules can be found in our recent work [37]. In the implemented trigger-action programming platforms, recommendations are continuously adapted in real-time to the current personalization goal of the user, and can be used to suggest relevant actions to "auto-complete" a defined trigger.

Chapter 5

End-User Debugging in Trigger-Action Programming

Besides supporting users in defining and discovering IF-THEN rules, another important and urgent challenge is the need to avoid possible *conflicts* [21] and to assess the *correctness* of IF-THEN rule [46] (*run-time problems* issue). Problems in trigger-action programs, indeed, negatively influence users' ability to correctly predict the outcomes of trigger-action programs [14], and can lead to unpredictable and dangerous behaviors [17], e.g., a door that is unexpectedly unlocked. Unfortunately, contemporary trigger-action programming platforms often expose too much functionality [71] and adopt technology-dependent representation models, thus forcing users to have a deep knowledge of all the smart devices and online services involved. As a result, users frequently misinterpret the behavior of trigger-action rules [17], often deviating from their actual semantics, and are prone to introduce errors [70].

Previous work started to address the challenge of avoiding run-time problems in trigger-action programming leveraging software engineering techniques, e.g., formal verification [86, 154] and information flow-control [133]. Instead of checking rules *"off-line"*, i.e., after their definition, we agree with previous work [14] that users must be able to identify programming bugs in IF-THEN rules and reason about how to fix them *during* the definition process. This implies designing tools for trigger-action programming that are specifically tailored for end users who are not accustomed to programming, by providing users with mechanisms to *debug* their IF-THEN rules.

Debugging is the process of finding the cause of an identified misbehavior and fixing or removing it. If prompted with the right information, even end users are able to design correct applications and programs [24]. To investigate end-user debugging in the trigger-action programming context, we firstly reviewed previous works on rule analysis in different areas to understand how to characterize, model, and detect problems in IF-THEN rules (Section 5.1). Then, we proposed 2 different end-user debugging tools:

a) *EUDebug* (Section 5.2), an integration of different end-user debugging features on top of an IFTTT-like interface; and,

b) *My IoT Puzzle* (Section 5.3), a tool to define and debug IF-THEN rules based on the Jigsaw metaphor.

Part of the work described in this chapter has been previously published in three different papers. The description and the evaluation of *EUDdebug*, in particular, is based on the work published in [44] and [33], while *My IoT Puzzle* was initially presented in [35].

5.1 Run-Time Problems in IF-THEN Rules

To better understand which problems should be detected and shown to end users in trigger-action programming platforms, we reviewed previous works on rule analysis in different contexts, e.g., [85, 139, 99]. Then, we defined a novel Semantic Colored Petri Net (SCPN) formalism to model and check IF-THEN rules at definition time.

5.1.1 Background

Many prior works faced the problem of formally or semi-formally verifying event-based rules with different approaches, especially in the area of databases [59, 85], expert systems [146], and smart environments [139, 5]. Rules, indeed, have the ability to interact with each other, and even a small set of dependencies between them makes it hard (and often undecidable) the problem of predicting their overall behavior [6].

In the field of smart environments, Vannucchi et al. [139] adopted formal verification methods for ECA rules, while Augusto and Hornos [5] proposed a methodological guide to use the Spin model checker to inform the development of more reliable intelligent environments.

Li et al. [85], instead, proposed a Conditional Colored Petri Net (CCPN) formalism to model and simulate Event-Condition-Action (ECA) rules for active databases. Petri nets were also used by Yang et al. [146] to verify rules in expert systems, and by Jin et al. [74] to dynamically verify ECA properties such as termination and confluence. Petri nets are bipartite directed graphs, in which directed arcs connect places and transitions. Places may hold tokens, which are used to study the dynamic behavior of the net. They can naturally describe the rules as well as their non-deterministic concurrent environment [74]. In our work, we defined a novel Petri net formalism, similar to CCPN but enhanced with new elements and with semantic information. Furthermore, different to the majority of the works described above, that aim at checking "off-line" the consistency of a set

of fixed and *already* defined rules, our goal was to assist end users in real time, i.e., *during* the actual definition of IF-THEN rules.

5.1.2 Characterizing Problems

Recently, Brackenbur et al. [14] identified ten programming bugs that might arise in IF-THEN rules by reviewing existing literature, and by considering bugs from other domains. They found, in particular, a) bugs in control flow, b) timing-related bugs, and c) errors in user interpretation. Since we aim at assisting users during the definition process, we focused on control-flow problems, i.e., *loops*, *inconsistencies*, and *redundancies*. In the following formal definitions of the problems, $R(T, A)$ models an IF-THEN rule where T is the trigger and A is the action. Furthermore, the notation e_x is used to describe a specific connected entity, i.e., a smart device or an online service, that belongs to the set E of all the supported connected entities.

Loops. Loops occur when a set of trigger-action rules are continuously activated without reaching a stable state [22, 99]. More formally, given a set S of TA rules, a loop arise when it exists at least a subset $S' \subset S$ whose cardinality is $|S'| \geq 2$ such that $\forall(R_i, R_{i+1}) \in S'$ the following conditions are met:

1. $A_i \Rightarrow T_{i+1}, 1 \leq i < n - 1$;
2. $A_n \Rightarrow T_1$.

Here, the implication symbol means that an action of a rule activates the trigger of another rule of the same set, and the index i models the run-time execution order of the n trigger-action rules, i.e., the order in which the rules are triggered by each other. An example of a loop is:

- if I post a photo on *Facebook*, then save the photo on my *iOS* library;
- if I add a new photo on my *iOS* library, then post the photo on *Instagram*;
- if I post a photo on *Instagram*, then post the photo on *Facebook*.

Inconsistencies. In active databases [85] and smart environments [22], inconsistencies occur when the execution order of rules may render different final states in the system. We generalized this concept to take into account all the elements of the contemporary IoT ecosystem, i.e., not only physical devices but online services, too. The large vision of the IoT ecosystem, that nowadays includes many different devices and online services in many different contexts, is intrinsically different from other domains. The order of actions performed on online services, e.g., posting a content on Facebook or sending a WhatsApp message, indeed, is not really important, because they do not change the internal state of a device and they do not leave the system in

a unpredictable or dangerous state. For this reason, inconsistencies in IF-THEN rules occur when rules that are activated at (nearly) the same time[1] may try to execute contradictory actions. More formally, given a subset of trigger-action rules $R_i(T_i, A_i)$, $1 \leq i \leq n$, an inconsistency arise if:

1. all the R_i are simultaneously or consequently executed, i.e.,
   - $T_i = T_j, 1 \leq i, j \leq n, i \neq j$; or
   - $A_i \Rightarrow T_j, 1 \leq i, j \leq n, i \neq j$;
2. $\exists A_i, A_j : A_i \neg A_j$ and $A_i, A_j \in e_x$.

where e_x is a specific connected entity. Here, the negation symbol means that the two actions are contradictory in terms of final functionality. An example of a set of rules that produces an inconsistency is:

- if my *Android* GPS detects that I exit the home area, then lock the *SmartThings* entrance door;
- if my *Android* GPS detects that I exit the home area, then set the *Nest* thermostat to Away mode;
- if the *SmartThings* entrance door is locked, then set the *Nest* thermostat to Manual mode.

Here, the three rules are executed at the same time because the first two rules share the same trigger, while the first rule implicitly activates the third rule. They produce two inconsistent actions, since they set 2 contradictory modes on the Nest thermostat, i.e., Away and Manual.

Redundancies. Redundancies occur when two or more rules that are activated (nearly) at the same time have replicated functionality[22]. Given a subset of trigger-action rules $R_i(T_i, A_i)$, $1 \leq i \leq n$, there is a redundancy when:

1. R_i are simultaneously or consequently executed, i.e.,
   - $T_i = T_j, 1 \leq i, j \leq n, i \neq j$; or
   - $A_i \Rightarrow T_j, 1 \leq i, j \leq n, i \neq j$;
2. $\exists\ A_i, A_j : A_i$ is similar to A_j, i.e.,
   - $A_i = A_j$ and $A_i, A_j \in E_x$; or
   - A_i and A_j provide the same functionality.

An example of a set of rules that produce a redundancy is:

- if I play a new song on my *Amazon Alexa*, then post a tweet on *Twitter*;
- if I play a new song on my *Amazon Alexa*, then save the track on *Spotify*;
- if I save a track on *Spotify*, then post a tweet on *Twitter*.

[1] e.g., when rules share the same trigger or when some rules trigger other rules

Here, the three rules are executed at the same time because the first two rules share the same trigger, while the second rule implicitly activates the third rule. They produce two redundant actions, i.e., the first and the third rule post the same content on Twitter.

5.1.3 Modeling and Detecting Problems

To model and check loops, inconsistencies, and redundancies in IF-THEN rules, we defined a novel Semantic Colored Petri Net (SCPN) formalism. Petri nets were selected as they can naturally describe the rules as well as their non-deterministic concurrent environment [74], and they easily allow the step-by-step simulation of the run-time behaviors of the modeled rules: by firing a transition at a time, tokens move in the net by giving the idea of a possible execution flow. As a member of Petri nets family, Colored Petri Net (CPN) [73] combine the strengths of ordinary Petri nets with the strengths of a high-level programming language.

SCPN is a Colored Petri Net similar to the Conditional Colored Petri Net (CCPN) formalism [85] proposed to model ECA rules in active databases. Different from such a formalism, SCPN do not consider conditions, and it uses a semantic model both to generate and analyze the net. Furthermore, as explained in the following, each token assumes different semantic "colors" by moving in the net. Such semantic information allows the inference of more information from the simulation of the net, i.e., to discriminate between problematic and safe rules.

Adding Semantics to Trigger-Action Rules

The novel characteristic of the SCPN formalism is the usage of Semantic Web technologies in conjunction with a Colored Petri Net. Adding semantics to IoT objects, triggers, and actions is a common approach [3]. In our case, we exploited the IFTTT-translation version of *EUPont* (Section 3.1.3) as the semantic model, by adding further relationships between the modeled IFTTT triggers and actions, e.g., the fact that the action *turn on the Philips Hue lamp* implicitly activates the trigger *the Philips Hue lamp has been turned on*.

The SCPN formalism, in particular, exploits *EUPont* information to "color" the places of the Petri net and to discriminate between similar and contradictory actions in terms of final functionality, i.e., to detect inconsistencies and redundancies among rules, respectively. Actions that are classified under the same *EUPont* classes, indeed, are similar in terms of final functionality, while actions that do not share any *EUPont* class are functionally contradictory. For example, the two actions *"set the Nest thermostat to Home mode"* and *"set 25 Celsius degree on the Nest thermostat"* share the same final functionality, because they are both classified under the same *EUPont* class, i.e., `Increase Temperature`. Compared to these actions, the action *"set the Nest thermostat to Away mode"* is contradictory in

terms of functionality, because it is classified under a different *EUPont* class, i.e., `Lower Temperature`.

Formalism

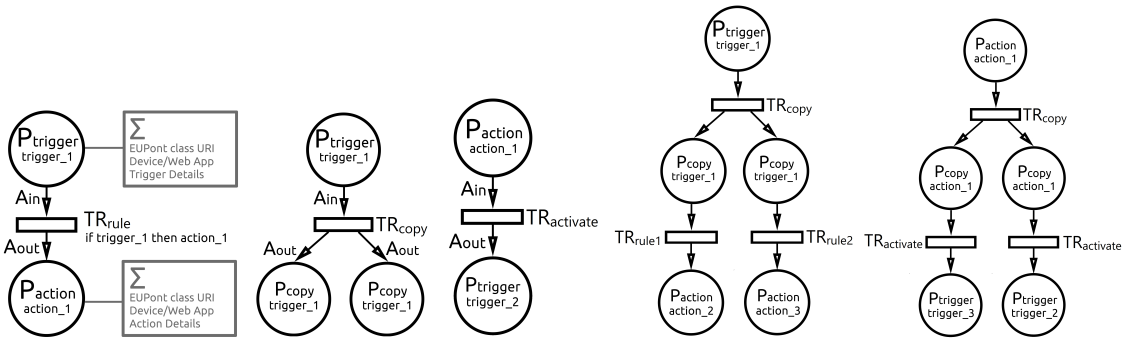


Figure 5.1: Figure 5.1a, Figure 5.1b, and Figure 5.1c show the basic elements of the SCPN formalism, while Figure 5.1d and Figure 5.1e show the 2 cases in which places are duplicated, i.e., when the same trigger is in common between two or more rules (Figure 5.1d), or when the same action implicitly activate two or more different triggers belonging to other rules (Figure 5.1e).

A SCPN is a 9-tuple

$$SCPN = \{P, T, \Sigma, A, N, C, G, E, I\} \tag{5.1}$$

where:

1. $P = T \cup A \cup T_{copy} \cup A_{copy}$ is a finite set of *places*. Given a set of IF-THEN rules, triggers and actions are modeled as T and A places, respectively (Figure 5.1a). Therefore, a token in a T place means that the associated trigger has been detected, while a token in a A place means that the associated action has been executed. T_{copy} and A_{copy} places are used when the same trigger is in common between two or more rules (Figure 5.1d), or when the same action implicitly activate two or more different triggers belonging to other rules (Figure 5.1e). In this case, the "original" place is replicated in such a way all the involved rules can be enabled when the trigger is detected, or the action is executed.
2. $TR = TR_{rule} \cup TR_{activate} \cup TR_{copy}$ is a finite set of *transitions*. A $TR = TR_{rule}$ models a connection from a T place to a A place by means of a IF-THEN rule, i.e., it models a rule defined by the user (Figure 5.1a). A $TR_{activate}$ models a connection from a A place to a T place and is used when an action of a rule implicitly activate the trigger of another rule (Figure 5.1c). Such transitions are extracted thanks to the semantic information of the EUPont

ontology. A TR_{copy} is used to connect a T place (or a A place) with its copies (Figure 5.1b).

3. Σ is a finite set of non-empty types, called *color* sets. Each Σ represent the color of a place, and contains the URI of the first *EUPont* class in the tree under which the associated trigger or action is classified, i.e., its final functionality, the smart device or online service by which the trigger or the action is offered, and all the details of the trigger or the action (Figure 5.1a).
4. A is a finite set of *arcs* such that $P \cap TR = P \cap A = TR \cap A = \emptyset$.
5. $N : A \rightarrow P \times TR \cup TR \times P$ is a *node* function.
6. $C : P \rightarrow \Sigma$ is a *color* function.
7. G is a guard function. In SCPN, the guard function returns true, always, i.e., $\forall t \in TR : [Type(G(t)) = True]$. Therefore, when there are enough tokens in the input places, the associated transition is fired, and a new token is generated in each output places.
8. E is an *arc expression* function that assigns an arc expression to each arc a such that $Type[E(a)] = C(p)_{MS}$, where p is the places connected to arc a.
9. I is an *initialization function* that assigns an initialization expression to each place p such that $Type[I(p)] = C(p)_{MS}$.

To exemplify and better explain the SCPN formalism, Figure 5.2 shows the net built starting from the seven rules in Table 5.1, with R7 being the rule in definition.

#	**Trigger (if...)**	**Action (then...)**
R1	my Android GPS detects that I exit the home area (**T1**)	lock the SmartThings entrance door (**A1**)
R2	the SmartThings entrance door is locked (**T2**)	set the Nest thermostat to Away mode (**A2**)
R3	the SmartThings entrance door is locked (**T2**)	turn off the Philips Hue lamp in the kitchen (**A3**)
R4	the SmartThings entrance door is unlocked (**T3**)	arm the Homeboy security camera (**A4**)
R5	the Homeboy security camera is armed (**T4**)	send me a Telegram message (**A5**)
R6	the Homeboy security camera is armed (**T4**)	lock the SmartThings entrance door (**A1**)
R7	the Homeboy security camera is armed (**T4**)	unlock the SmartThings entrance door (**A6**)

Table 5.1: The rules that generate the SCPN of Figure 5.2.

Triggers and actions of a given connected entity, e.g., *SmartThings*, are modeled as T and A places, respectively, e.g., $T1$ and $A1$. When a trigger is in common between more than one rule, as in R2 and R3, the associated places are duplicated (e.g., $T2_{copy}$ in Figure 5.2) and connected through a TR_{copy} transition. When a

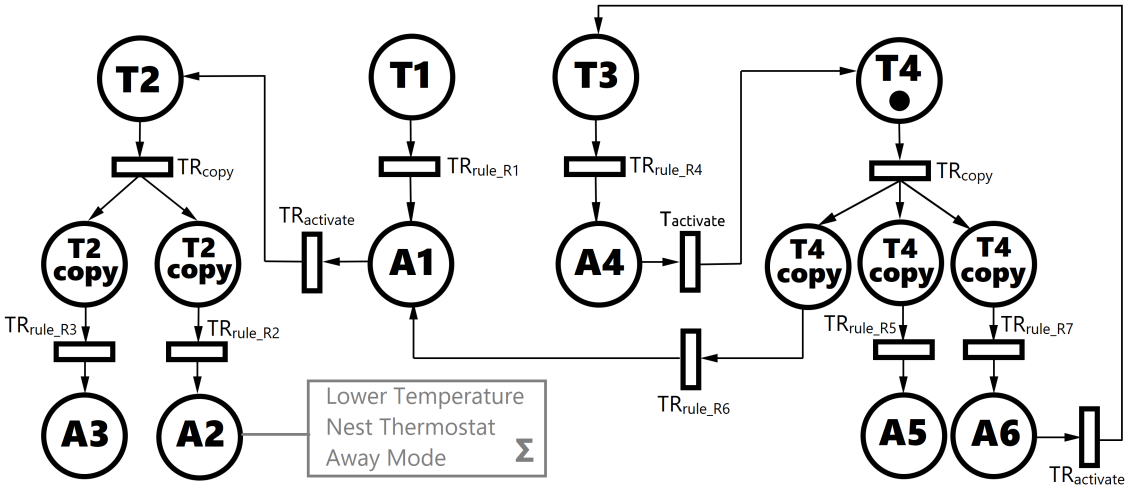


Figure 5.2: The SCPN generated by analyzing the rules of Table 5.1.

token is in the original place, such a transition simply replicates the token in each copied place. A places follow a slightly different process than T places: an A place can be reused for rules that share the same action, e.g., $A1$ models the action offered by both R1 and R6 (Figure 5.2). The rule transitions, e.g., $TR_{rule_R}2$, model the 7 rules of Table 5.1 by connecting the involved triggers and actions. The activate transitions, instead, e.g., the one between $A1$ and $T2$, model the fact that an action of a rule triggers the event of another rule. Finally, as exemplified for A2 in Figure 5.2, all the places are characterized by a semantic color that represents the semantic information associated with the corresponding trigger or action. When a token cross a place, it assumes the place color.

Rule Analysis with SCPN

To detect loops, inconsistencies, and redundancies in IF-THEN rules at definition time, the following procedure is adopted:

Net Generation. Given a set of IF-THEN rules already defined, and the "current" rule, i.e., the rule that is being defined, the first step of the rule analysis is the generation of a SCPN net. All the IF-THEN rules are firstly translated in the *EUPont* semantic representation. Triggers and actions are enumerated in order to create a T and a A for each unique trigger and action, respectively. Furthermore, thanks to the semantic information extracted from *EUPont*, a semantic color is assigned to each place. For each trigger in common between two or more rules, and for each action that implicitly activate two or more different triggers belonging to other rules, a series of T_{copy} and A_{copy} are created. Copy places are linked to the corresponding T or A places by means of TR_{copy} transitions. By analyzing the set of IF-THEN rules, then, trigger

places, either "original" (T) or copies (T_{copy}), are connected to action places (A) by means of TR_{rule} transitions. Furthermore, by reasoning on the information extracted from the *EUPont* ontology, further transitions are added to model rules that implicitly activate other rules. In this case, action places, either "original" (A) or copies (A_{copy}), are connected to trigger places (T) by means of $TR_{activate}$ transitions.

Initial Marking. After the generation, the net is marked with an initial marking. Different strategies can be adopted: when the net is used to assist users during the rule definition process, in particular, the initial marking is a single token in the T place related to the rule that is being defined.

Net Analysis. When the net is marked, it can be analyzed. Loops are detected by performing a depth-first search on the net. To detect inconsistencies and redundancies, instead, the SCPN is executed, and all the A places crossed by the tokens during the execution, i.e., the executed actions, along with the associated semantic colors are analyzed. An inconsistency is found if there are at least two executed actions that a) act on the same connected entity, and b) are classified under different *EUPont* classes. Similarly, a redundancy is found if there are at least two executed actions that a) act on the same connected entity, and b) share the same *EUPont* classes.

Net Simulation. Beside being analyzed, the net can be executed *step-by-step* by randomly selecting a transition to be fired from the set of transitions that are enabled in a given moment.

As an example, the net in Figure 5.2 presents a loop arising between R4 and R7. A redundancy and an inconsistency are also present, and can be identified by executing and analyzing the net. The net is initially marked with a token a single token in the T place related to the rule that is being defined, i.e., $T4$ for R7. Then, the net is executed, and the activated transitions move the token in the net. When the token is in a T place, all the rules that share such a specific trigger are activated: in the first step of the execution of Figure 5.2, for example, the token is removed from $T4$ by the TR_{copy} and replicated in each copy of $T4$, thus enabling the TR_{rule} of $R6$, $R5$, and $R7$. In the next step, the net may execute $R6$, i.e., one of the activated TR_{rule}, thus moving the token from the $T4_{copy}$ to $A1$. This simulate the execution of action associated with $A1$.

Following this process, the analysis of the crossed A places show that a redundancy arises since the action of R5 ($A5$) contains multiple tokens at the end of the net execution, thus generating many Telegram messages (an infinite number), as the trigger of R5 ($T4$) is involved in the loop. Crossed places also highlight an inconsistency, $A1$ and $A6$ model two inconsistent actions, i.e., "lock the entrance door" and "unlock the entrance door". By getting rid of R7, a user can eliminate all those problems.

5.1.4 SCPN RESTful Server

We implemented the SCPN formalism in a RESTful server by exploiting the Java Spring framework[2] (Figure 5.3). The server is composed of three modules: *Rule Service*, *SCPN Service*, and *Rule Controller*. The Rule Service offers the features needed to manage collections of IF-THEN rules, i.e., to create, read, update, and delete rules through the interaction with a MySQL database. Once a rule is saved, the SCPN Service generates and analyzes the SCPN by retrieving the defined rules from the Rule Service, and by using the OWL API[3] library to extract the needed semantic information from the *EUPont* ontology. The same module is also responsible for the step-by-step simulation of the involved rules. Finally, the Rule Controller exposes a list of REST APIs that can be exploited by a user interface, as shown in the following two sections describing the *EUDdebug* and *My IoT Puzzle* tools, respectively.

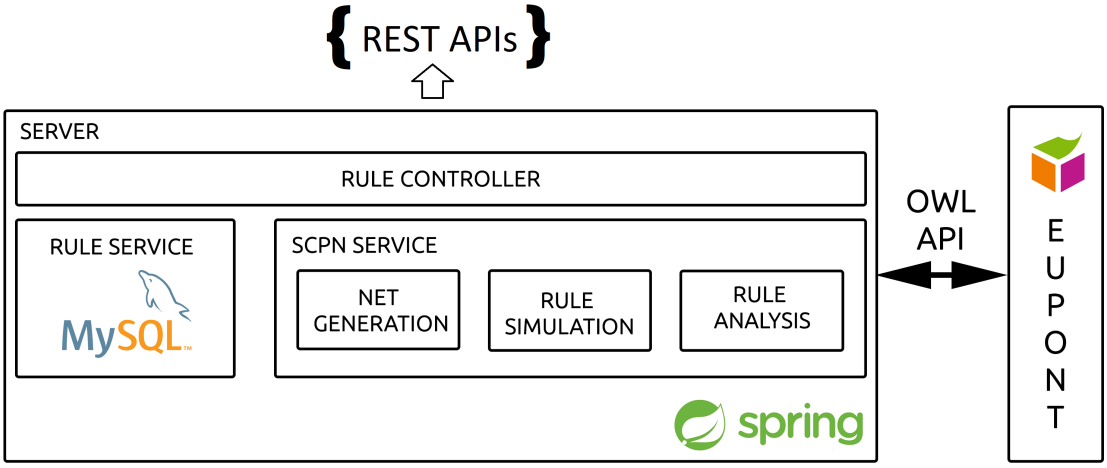


Figure 5.3: The SCPN RESTful server architecture.

5.2 Exploring End-User Debugging in Trigger-Action Programming Platforms

To firstly explore whether end users would be able to debug their IF-THEN rules at definition time, we proposed *EUDebug*, a system that exploits the same definition interface, metaphors and expressiveness of IFTTT, and that enables end

[2] https://spring.io, last visited on November 12, 2019

[3] http://owlapi.sourceforge.net, last visited November 12, 2019

users to debug their IF-THEN rules according to two strategies: (i) by assisting them in identifying rule conflicts, and (ii) by helping them foresee the run-time behavior of their rules through step-by-step simulation. Figure 5.4 shows a sample usage scenario:

a) The user defines a new IF-THEN rule in a web-based application modeled after IFTTT (e.g., "if the security camera in the office is armed, then unlock the door"). As the user is defining the rule, *EUDebug* employs SCPN to model, check, and simulate the rule with respect to previously defined IF-THEN rules.

b) When the rule definition is completed, *EUDebug* highlights possible problems that the rule may generate, by providing a short explanation to the user.

c) If needed, the user can further inspect and understand the problems by asking *EUDebug* to perform and show a step-by-step simulation of the problematic rules.

d) Finally, the user can edit the defined rule or decide to ignore the highlighted problems, thus saving the rule in the current format.

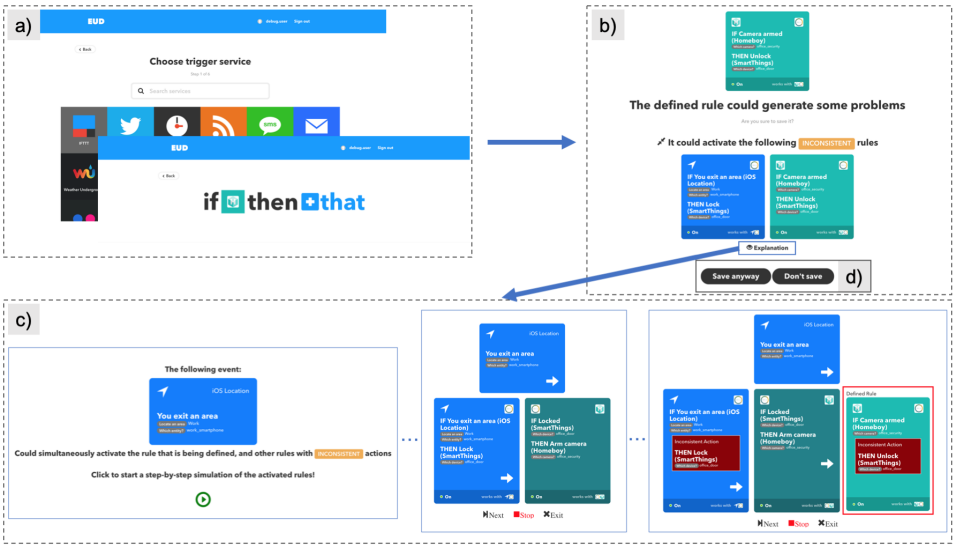


Figure 5.4: *EUDebug* is a system for debugging IF-THEN rules that allows the user to: a) define a new rule; b) view any problems that the rule may generate; c) further investigate each problem with a step-by-step simulation; and d) edit the rule to fix the problem or save it anyway.

5.2.1 Background

Today's end-user programmers include anyone who creates artifacts that instruct computers how to perform an upcoming computation, without being necessarily interested in learning how to program. Along with the ability to create programs comes the need to debug them, and work on end-user debugging is only beginning to become established [63].

To our knowledge, *EUDebug* is one of the first attempt to provide debugging features to end users who define IF-THEN rules for their IoT ecosystem. Previous work on end-user debugging are not related to trigger-action programming nor the IoT, but mainly focus on mashup programming [24] and, especially, spreadsheets [18, 76, 63]. In using spreadsheets, in particular, users are likely to make a large number of mistakes [19, 114], and the need of supporting end-users' debugging efforts in such tools has gained interest in the last years [63]. By focusing on novice developers, instead, Ko and Myers [77] propose an interrogative debugging interface for the Alice programming environment. The interrogative debugging is a debugging paradigm by which novices can ask *why* and *why not* questions about their program's run-time failures. *EUDebug* is inspired by this paradigm, as it assists end users in understanding *why* their trigger-action rules may be problematic through the step-by-step simulation of the run-time behavior of the rules. To model, check, and simulate IF-THEN rules, in particular, *EUDebug* exploits the SCPN formalism described in Section 5.1.3.

5.2.2 The *EUDebug* Tool

We developed *EUDebug* as a web-based end-user debugging interface through the Angular framework[4]. The tool exploits the SCPN RESTful server (Section 5.1.4) to assist users during the definition of IF-THEN rules. The interface can be logically split in three parts: a) *Rule Definition*, b) *Problem Checking*, and c) *Step-by-Step Explanation*.

The Problem Checking and the Step-by-Step Explanation interfaces implement two strategies for end-user debugging, respectively: identification of rule conflicts, and simulation of the run-time behavior. To allow the definition of IF-THEN rules, we modeled the definition interface after IFTTT (Figure 5.5) due to the popularity of the platform [46], its ease of use and accuracy in the rule definition process [21], and the availability of real usage data [137], which we used to define available triggers and actions. In addition, the form-filling procedure it adopt helps users to avoid syntactical errors during the rule definition. As reported in Section 1.1.1, to define a rule a user needs to first select which connected entity they want to use as a trigger (Figure 5.5). Once they select an entity, they can choose the specific

[4] `https://angular.io`, last visited on September 18, 2018

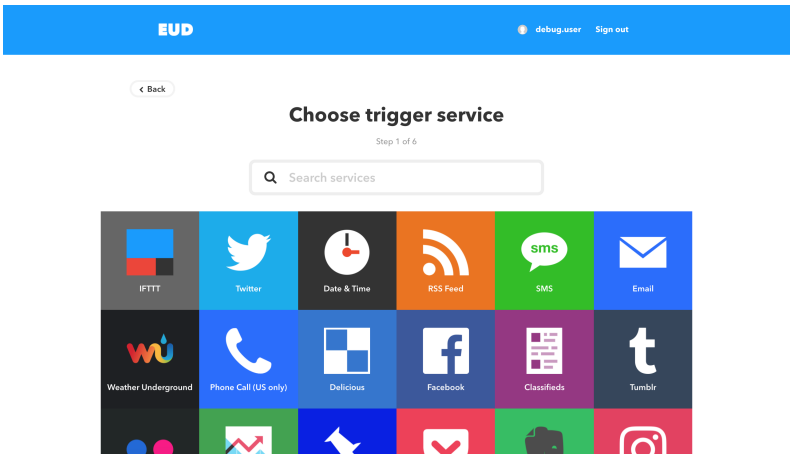


Figure 5.5: The user interface for defining a new IF-THEN rule, showing the selection of the connected entity to be used as a trigger.

trigger to be used (e.g., "turned on" for *Philips Hue* lamps) and fill any additional information required by the trigger (e.g., which *Philips Hue* lamp they want to use). To define the action part of the rule, the user has to repeat the same steps.

The defined rule is, then, described according to the SCPN formalism, and analyzed by the net to look for any loops, inconsistencies, and redundancies. The results of the analysis of the SCPN are, in real time, shown to the user in the Problem Checking interface (Figure 5.6). The *Problem Checking* interface shows the rule just defined by the user and any problems that the rule may generate. In Figure 5.6, for instance, a possible inconsistency between two rules is highlighted. To better understand the problems and to foresee the run-time behavior of the involved trigger-action rules, the user can click on the "Explanation" button to open the *Step-by-Step Explanation* interface (Figure 5.7). In such an interface, the user can simulate step-by-step what happens within their rule, to try to understand why the highlighted problems arise. For instance, Figure 5.7 shows that the event *"You exit an area"* activates a sequence of IF-THEN rules that includes the rule that is being defined, and two inconsistent actions that close and open a door at the same time.

5.2.3 User Evaluation

We ran an exploratory study with 15 participants to evaluate whether *EUDebug* helps them a) *understand* and b) *identify* problems that may arise in their IF-THEN rules. The following questions guided our study:

RQ9) Can *EUDebug* help end users debug their IF-THEN rules? Do they understand the involved problems and why their rules generate them?

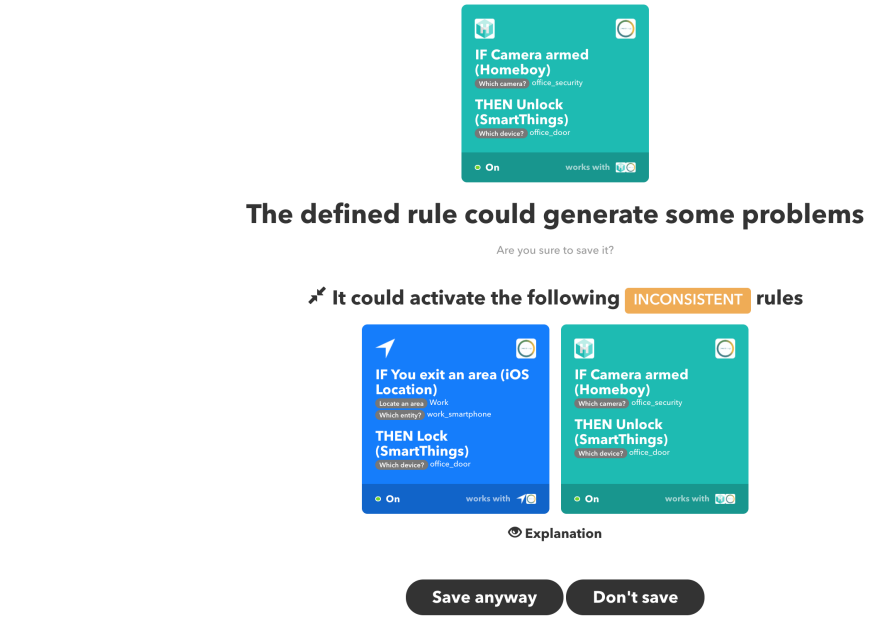
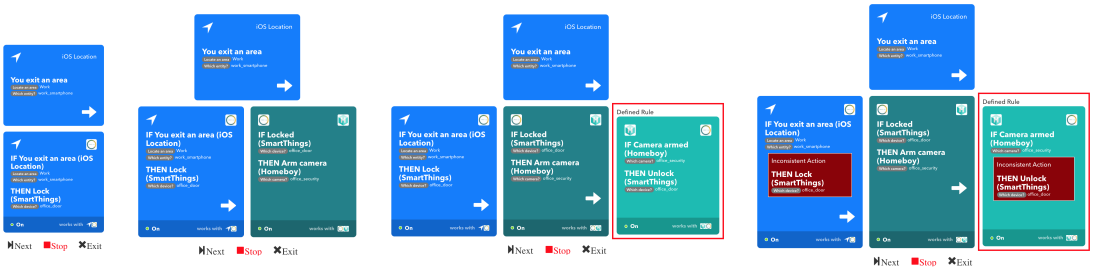


Figure 5.6: The Problem Checking interface showing an inconsistency between an already existent rule and the defined one.



(a) (b) (c) (d)

Figure 5.7: Step-by-Step Explanation: a sequence of screenshots of the user interface related to the step-by-step simulation of the inconsistency problem of Figure 5.6.

RQ10) Is highlighting the detected problems sufficient to identify such problems, or do users need the additional details provided by the step-by-step simulation?

Participants

We recruited 15 university students (9 males and 6 females) with a mean age of 20.34 years ($SD = 2.50$, $range : 18 - 25$) by sending emails and private messages to students coming from various backgrounds. We excluded users who had previous experience in computer science and programming. Table 5.2 summarizes the

demographics of our participants. On a Likert-scale from 1 (Very Low) to 5 (Very High), participants stated their level of technophilia ($M = 3.94$, $SD = 0.80$) and technological savviness ($M = 2.67$, $SD = 0.82$). Furthermore, on a Likert-scale from 1 (No knowledge at all) to 5 (Expert), participants declared their experience with trigger-action programming ($M = 1.34$, $SD = 1.04$).

Table 5.2: General demographics in the *EUDebug* user study.

Id	Age	Gender	Education	Technophilia	Tech savviness	TA Programming
P1	21	F	Education sciences	3	2	1
P2	19	M	Aerospace engineering	5	4	2
P3	19	M	Aerospace engineering	5	4	5
P4	19	M	Management	4	2	1
P5	20	M	Biology	5	3	1
P6	22	F	Law	4	3	1
P7	25	M	Architecture	4	3	1
P8	24	F	Music therapy	3	3	1
P9	25	M	Civil engineering	4	2	1
P10	18	F	Scientific high school	4	3	1
P11	19	F	Linguistic high school	3	1	1
P12	18	M	Agricultural high school	3	2	1
P13	19	M	Economics	5	3	1
P14	18	M	Industrial Institute	4	3	1
P15	19	F	Architecture	3	2	1

Procedure

We brought each participant to our lab for a 45-minute session using *EUDebug* on a Macbook Pro connected to an external 22-inch monitor. At the beginning of the study, participants were introduced to trigger-action programming and to *EUDebug* with an example of a rule definition. To allow us to investigate the *EUDebug* understandability (**RQ9**), participants were not introduced to the problems that rules may generate. We then presented a task requiring the definition of the 12 trigger-action rules reported in Table 5.3, which include both smart devices and online services. The rules generated 5 different problems (i.e., 2 inconsistencies, 2 redundancies, and 1 loop):

- **IC1**. TA1 and TA2 generate an **inconsistency**, because they share the same trigger ("entering the home area") while producing contradictory actions on the same device, i.e., they turn on and off the same *Philips Hue* lamp, respectively;
- **IC2**. TA7 and TA9 generate an **inconsistency**, because they produce contradictory actions on the same device, i.e., locking and unlocking the office

ID	Trigger Service	Trigger	Action Service	Action
TA1	Android Location	You enter an area (*where:* home)	Philips Hue	Turn on lights (*what:* kitchen lamp)
TA2	Android Location	You enter an area (*where:* home)	Philips Hue	Turn off lights (*what:* kitchen lamp)
TA3	Android Location	You enter an area (*where:* home)	Philips Hue	Turn on color loop (*what:* kitchen lamp)
TA4	iOS Photo	New photo added to album (*album:* ios photos)	Dropbox	Add file from URL (*URL:* ios photo, *folder:* drpb photos)
TA5	Dropbox	New file in your folder (*Folder:* drpb photos)	Facebook	Upload a photo from URL (*URL:* drpb photo)
TA6	Facebook	New photo post by you	iOS Photo	Add photo to album (*URL:* facebook photo, *Album:* ios photos)
TA7	iOS Location	You exit an area (*where:* work)	SmartThings	Lock (*what:* office door)
TA8	SmartThings	Locked (*what:* office door)	Homeboy	Arm camera (*what:* office camera)
TA9	Homeboy	Camera armed (*what:* office camera)	SmarThings	Unlock (*what:* office door)
TA10	Amazon Alexa	New song played	Twitter	Post a tweet (*text:* I liked the Alexa song)
TA11	Amazon Alexa	New song played	Spotify	Save a track (*track:* alexa song)
TA12	Spotify	New saved track	Twitter	Post a tweet (*text:* I liked the Spotify song)

Table 5.3: The 12 trigger-action rules defined in the *EUDebug* study.

door, and they are activated nearly at the same time, since TA7 activates TA8, and TA8 activates TA9. Indeed, the action of TA7 ("lock the office door") activates the trigger of TA8 ("the office door has been locked"), while the action of TA8 ("arm the office camera") activates the trigger of TA9 ("the office camera has been armed");

- **RD1**. TA1 and TA3 generate a **redundancy**, because they share the same trigger ("entering the home area") while producing two similar actions on the same device, i.e., they turn on the same *Philips Hue* lamp with different colors;

- **RD2**. TA10 and TA12 generate a **redundancy**, because they produce similar actions on the same online service ("posting a tweet about a liked song") and are activated nearly at the same time, since TA10 and TA11 share the same trigger ("new song played on Alexa") and TA11 activates TA12. Indeed, the action of TA11 ("save the song on Spotify") activates the trigger of TA12

("new song saved on Spotify");

- **LP**. TA4, TA5, and TA6 generate an infinite **loop**, because TA4 activates TA5, TA5 activates TA6, and TA6 activates TA4. Indeed, the action of TA4 ("add file to Dropbox") activates the trigger of TA5 ("new file added to Dropbox"), the action of TA5 ("upload a photo on Facebook") activates the trigger of TA6 ("new photo post on Facebook"), and the action of TA6 ("add a photo to iOS Photo") recursively activates the trigger of TA4 ("new photo added to the iOS Photo library").

Rules were presented one at a time on a sheet of paper in a counterbalanced order. To make sure that all the participants experienced a given problem in the same way, however, we maintained the order within each problem, e.g., TA2 was always presented after TA1. When the *EUDebug* interface highlighted some problems (Problem Checking), participants were free to decide whether to save the rule or not. Our aim was to investigate whether participants understood the presented problems and their dangerousness, without forcing them to discard problematic rules. Before deciding, participants could optionally use the Explanation button to perform the step-by-step simulation of the rules that generated the problem. All the sessions were audio recorded for further analysis.

Measures

During the study, we collected the following quantitative measures:

- **S:** number of rules that generated a problem *saved* anyway by participants, monitored for each highlighted problem, e.g., number of saved rules in case of loops.
- **D:** number of rules that generated a problem *discarded* by participants, monitored for each highlighted problem.
- **SbS:** number of times participants used the *Step-by-Step Explanation* when experienced a specific problem.

In addition, if participants used the Step-by-Step Explanation, we asked them:

- **SbS Motivation:** why they decided to use the Step-by-Step Explanation.
- **SbS Usefulness:** whether and how the explanation helped (or not) them to understand the problem.

Furthermore, when the definition of a rule generated a problem, we asked participants for their **Interpretation**. The interpretation was asked *before* the optional usage of the Step-by-Step Explanation interface. In particular, when a participant

decided to *discard* a rule that generated a problem, they had to demonstrate their understanding of the problem by retrospectively explaining why the rule generated the issue. When they decided to *save* anyway a rule that generated a problem, instead, they had to justify their choice. In the next sections, we present and discuss the findings of the study, by organizing the discussion around the main topics that emerged from the analysis of the results. Qualitative analysis was conducted by two researchers in an iterative coding process.

5.2.4 Results

Differences in Users' Behavior

Most of the participants perceived *EUDebug* as a helper for understanding whether the highlighted problems were "dangerous" or not (**RQ9**). Moreover, they exhibited different behaviors when facing the various problems, i.e., they considered *redundancies* as less problematic than loops and inconsistencies, at least in some specific cases.

In detail, we analyzed how many times participants saved (or discarded) a rule that generated a given problem, i.e., the S and D measures. As reported in Table 5.4, 12 participants out of 15 (80%) discarded TA6, i.e., the rule that generated the loop L. Instead, participants discarded the rule that generated an inconsistency in the 96.67% of the cases, on average: for IC1, 14 participants out of 15 (93.34%) discarded TA2, while for IC2 all the participants discarded TA12. This seems to suggest that participants were aware of the "danger" caused by such problems. Conversely, participants discarded the rule that generated a redundancy, i.e., RD1 and RD2, only in 53.34% of cases, on average. Therefore, at least in some cases, redundancies seemed to be considered less "dangerous" and even acceptable than loops and inconsistencies.

Rule	Problem	Type	S	D	SbS
TA2	**IC1**	Inconsistency	1	14	5
TA9	**IC2**	Inconsistency	0	15	5
TA3	**RD1**	Redundancy	12	3	3
TA12	**RD2**	Redundancy	2	13	8
TA6	**LP**	Loop	3	12	7

Table 5.4: The number of times participants (N = 15) saved a rule (S), discarded a rule (D), or used the Step-by-Step explanation (SbS) when a problem is highlighted.

Since participants had opposite behaviors when facing with the two *redundancies*, we further analyze the collected data and the audio recording of the entire

session. In fact, only 3 participants out of 15 (20%) discarded the rule that generated the RD1 problem, while 13 participants out of 15 (86.67%) discarded TA12, i.e., the rule that generated RD2. The reason for such a difference in the participants' behavior can be glimpsed by inspecting the *nature of the rules* involved in the two redundancies. In the first redundancy, considered as "acceptable" by the majority of the participants, both involved rules turned on the kitchen lamp with different colors. Instead, the second redundancy, considered as "unacceptable" by the majority of the participants, produced two similar messages on Twitter. We can preliminary conclude that redundancies in the "virtual" world, e.g., multiple messages on the web, are more annoying compared to redundancies in the "physical" world. In fact, rules in "physical" redundancies often send similar commands to a device without drastically modifying its current state, e.g., the fact that a lamp is turned on. On the contrary, "virtual" redundancies typically result in duplicated messages and notifications, a potentially more annoying behavior.

Differences in Users' Interpretation

Most of the participants gave a correct *interpretation* about their choice of saving or discarding a problematic rule. However, not all the problems were equally understood, with *loops* being the most difficult problem to understand.

To investigate whether participants understood the meaning of the encountered problems and why they happened, we used the *SbS* measure and the participants' *interpretations* extracted from the audio recording. As reported in the following, the loop turned to be the most difficult problem to understand, and led participants to frequently use the Step-by-Step Explanation.

Inconsistencies. For what concerns IC1, all the 14 participants that discarded TA2 provided a sound interpretation. P1, for example, said *"the rules did not have any sense. They turned the lights on and off at the same time. The two commands (turn the lights on and turn the lights off) cannot be executed at the same time."* P7, beside explaining the problem, also identified a possible alternative: *"I would have modified the trigger: this rule is ok when you exit the home area."* Only 1 participant, the one that decided to save TA2, provided an incorrect interpretation of the problem even after using the Step-by-Step Explanation. In her interpretation, in particular, she said *"I do not trust the platform, I am sure that such two rules will never be activated at the same time."* The 15 interpretations collected for IC2 are also encouraging. 11 participants, in particular, provided a sound interpretation after discarding TA9, such as *"if the door is locked, the camera is armed, but when the camera is armed, this rule unlocked the door!"* or *"this rule will unlock the door when I leave the office: not good."* The remaining 4 participants immediately discarded TA9, but they provided a misinterpretation. In their interpretation,

in particular, they focused on the rule they were evaluating, only, rather than on the entire chain of rules that generated the problem, i.e., TA7, TA8, and TA9. P7, for example, said *"I did not save the rule because I want the door to remain closed"*, while P8 said *"if the camera is armed, the door must be closed."* A possible explanation can be found in their decision to discard the rule without using the Step-by-Step Explanation. On average, 5 participants out of 15 (33.34%) used the Step-by-Step Explanation (Table 5.4).

Redundancies. The number of wrong interpretations of redundancies was similar to the number of wrong interpretations of inconsistencies. For what concerns RD1, 13 participants out of 15 (86.67%) provided a sound interpretation. In particular, 11 participants out of 12 (91.67%) successfully provided an interpretation for their decision to save TA3 anyway. All of them declared that they were aware of what would happen, and that the highlighted issue was not a problem at all. P6 said that the color can be seen as a *"new feature"* of the first rule, while P7 asserted that *"the important thing is that the lamp is turned on, I do not care its color."* The only participant that provided a wrong interpretation was P15, the same participants that made an error for IC1. No one of the 12 participants that saved TA3 used the Step-by-Step Explanation. Instead, all the 3 participants that discarded TA3 used the Step-by-Step Explanation, and 2 of them provided a sound interpretation, while the other focused on TA3, only, by saying *"I do not want a colored light in the kitchen"*. Also for the second redundancy, i.e., RD2, no one of the 2 participants that saved TA12 anyway used the Step-by-Step Explanation, but all of them provided a sound interpretation. Instead, 11 of the remaining 13 participants (84.61%) that discarded TA12 successfully provided an interpretation. P1, for example, explained exactly what happened by saying *"When I listen to a song on Alexa, the defined rules post a tweet and save the track on Spotify. Now I'm defining a rule to post on Twitter when I saved a track on Spotify, but there is already a post on Twitter!"* The remaining 2 participants, even after using the Step-by-Step Explanation, focused on TA12, only, by saying, for example, *"it does not have any sense to post on Twitter the song you are listening"*. On average, participants used the Step-by-Step Explanation in 36.67% of cases (Table 5.4).

Loop. The loop LP led participants to make more errors in their interpretations. Since a loop can never be considered as "acceptable", all the 3 participants that saved TA6 failed in providing a correct interpretation. P13, for example, did not understand that the 3 involved rules would be executed infinite times, because she said *"I am sure that this problem will never occur with the rules I have defined. Moreover, such rules are useful, because the photo will be saved in 3 places at the same time."* Furthermore, also 3 of the participants that discarded TA6 provided an incorrect interpretation. The prevailing error was

that participants did not understand that the involved rules would have been executed for an infinite number of times: both P1 and P12, for example, said *"I did not save the rule because otherwise the same photo would have been shared twice on Facebook."* Therefore, in line with previous works [14], results suggest that a loop among IF-THEN rules is one of the most complex concept to understand. A series of paired-samples t-test confirm this finding. In fact, the number of errors in loop interpretations was significantly higher than in redundancies ($t(14) = 2.25$, $p < 0.05$), while such a difference was not significant with respect to inconsistencies ($t(14) = 1.97$, $p = 0.06$). For the loop, the Step-by-Step Explanation was more used (7 participants out of 15, 46.67%, Table 5.4) than for the other problems. A possible explanation of such understandability problems is that the concept of loop is strictly related to the mental model of users with a computer science background.

We also noticed a possible link between the "nature" of a problem and its understandability by further analyzing the number of *Step-by-Step Explanations* used and the number of wrong interpretations in each problem. In particular, when subjected to the first 2 problems, i.e., IC1 and RD1, participants used the Step-by-Step Explanation in fewer cases, and provided less wrong interpretations with respect to the other three problems, i.e., LP, IC2, and RD2. Such a difference can be associated with the nature of the problems. IC1 and RD1, in fact, are *direct* problems, i.e., problems between rules that shared the same trigger. On the contrary, LP, IC2, and RD2 are *indirect* problems, because they are caused by implicit activations between rules, i.e., an action of a rule that implicitly activates the trigger of another rule. Figure 5.8 visually shows the differences between direct and indirect problems, and further suggests that indirect problems are more difficult to understand, and need more efforts, e.g., a step-by-step simulation, to be identified by end users.

Highlighting Problems or Explaining Them?

To investigate whether participants found more useful one of the two strategies adopted by *EUDebug* for identifying problems in trigger-action rules (**RQ10**), we studied the correlation between the interface used, i.e., Problem Checking or Step-by-Step Explanation, and the participants' *interpretations* in case of a problem (Table 5.5). On average, the usage of the Problem Checking interface, only, resulted in a correct interpretation in 77.81% of cases. When participants decided to use the Step-by-Step Explanation, the percentage of correct interpretations increased to 83.78%. Such a difference is particularly evident for the loop L. Only 50% of the participants, in fact, discarded TA6 by providing a correct interpretation by using the Problem Checking interface, only. Participants that used the Step-by-Step Explanation, instead, provided a correct interpretation in 71.43% of cases. This

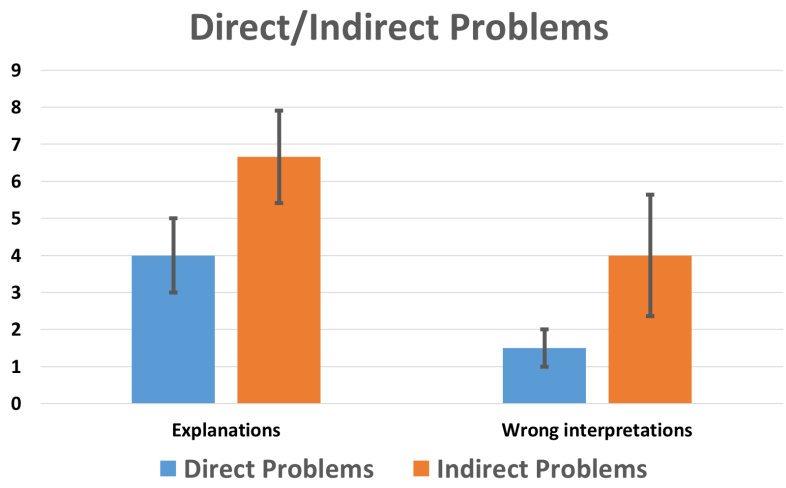


Figure 5.8: Average number of explanations used and average number of wrong interpretations for direct/indirect problems

seems to suggest that, at least in some cases, highlighting the detected problems (i.e., Problem Checking) may be *not* sufficient to allow end users in identifying possible problems in their rules, and that a step-by-step simulation of the involved rules could instead help users in understanding what happens. To confirm this finding, we analyzed the participants' feedback about the usage of the Step-by-Step Explanation (used 28 times in total) by group it into several topics described below.

	L	**I**	**R**	**Total**
PC success	50%	85%	84.52%	77.81%
SbS success	71.43%	80%	93.75%	83.78%

Table 5.5: Number of times participants provided a correct *interpretation* by using the Problem Checking interface, only (PC success), or after using the Step-by-Step explanation (SbS success).

For what concern the *SbS Motivation*, in most of the cases participants asserted that they used the Step-by-Step Explanation to "better understand the problem" (13). When subjected to IC1, for example, P1 said *"I used the Step-by-Step Explanation because I did not understand the problem. The two rules seemed the same to me."*. Similarly, P10 used the step-by-step explanation for RD2 *"to better understand the redundancy concept"*, while P15 provided the same motivation when subjected to the loop. In a considerable number of cases (8), the Step-by-Step Explanation was instead used because "the problems were composed of too many steps", i.e., rules that activated other rules. Not surprisingly, such motivation was

used for indirect problems, only. In one case, for example, P14 said *"I used the Explanation because I did not understand the execution path of the rules"*, while in another case, P12 said *"I used the Explanation because I did not understand the relationship between the rules."* In the remaining cases, participants used the Step-by-Step Explanation because "they did not remember a rule they defined before" (4), "to confirm their first idea about the problem" (2), and because the "Explanation helped them before" (1).

Participants provided interesting feedback also when asked to evaluate whether and how the Explanation helped (or not) them in understanding the problems (*SbS Usefulness*). In 13 cases, participants asserted that the Step-by-Step Explanation was useful because it allowed them "to see all the involved steps." Participants provided this feedback for indirect problems, mainly. The loop, in particular, was the problem for which this feedback was more common. P6, for example, said *"the Explanation helped me in understanding the loop because I could better see the evolution of the rules,"* while P10 pointed out that seeing the figures related to the rules one at a time helped her in understanding the problem. In other 5 cases, the Step-by-Step Explanation helped "participants to remember a rule they had defined before" (*"The Explanation helped me in understanding the problem because it told me: hey, you have defined this rule before!"*, P8). This feedback takes even more importance if we think to the real usage of an EUD platforms, where rules are defined in different moments, even months later. In other 5 cases, participants asserted that "the Explanation helped them by visually highlighting the problem" (*"The Explanation helped me to understand the problem because it visually told me what happened"*, P6). In the remaining cases, participants provided generic feedback about the usefulness of the Explanation, i.e., "it helped me in understanding the problem" (3) and "it confirmed my first idea" (2).

5.3 Debugging IF-THEN Rules Through the Jigsaw Metaphor

Besides proposing and evaluating *EUDebug*, we focused on additional questions in the field of end-user debugging in trigger-action programming platforms that are still underexplored. Which visual languages, for instance, are more appropriate for debugging rules, and which information do end users need to understand, identify, and correct errors? To answer these questions, we firstly conducted a literature analysis by reviewing previous work on end-user debugging in different contexts, with the aim of extracting design guidelines. Then, we used the extracted guidelines to implement *My IoT Puzzle*, a tool to define and debug IF-THEN rules based on the Jigsaw metaphor. The tool interactively assists users in the definition process by representing triggers and actions as complementary puzzle pieces, and by providing real-time feedback to test *on-the-fly* the correctness of the rule under

definition. Puzzle pieces, for example, deteriorate over time according to their usage (Figure 5.10), while the tool is able to warn users in case of conflicts, namely infinite loops, inconsistencies, and redundancies (Figure 5.11). Furthermore, the tool empowers end users in resolving problems through textual and graphical explanations. Following the interrogative debugging paradigm [77], for instance, the tool is able to answer questions such as *"why it is not working?"*, thus providing the user with a textual explanation of the detected problem (Figure 5.12).

5.3.1 Background & Adopted Technologies

In the trigger-action programming context, work on end-user debugging is still in its early stage. While the majority of previous studies focused on mashup programming [24], spreadsheets [63], and novice developers [77], only a few recent works, including our *EUDebug* and the ITAD tool [92], started addressing the problem of end-user debugging in the IoT. ITAD (Interactive Trigger-Action Debugging), in particular, warns users in case of rule conflicts, and it allows the simulation of trigger-action rules in fixed contexts.

In this work, we stemmed from both *EUDebug* and ITAD for taking a step forward: with *My IoT Puzzle*, our aim was to understand how we could make debug of IF-THEN rules more *understandable* by end users. To identify problems between the defined rules, we used the same approach also adopted by *EUDebug* , i.e., our novel SCPN formalism.

5.3.2 Extracting Design Guidelines

We reviewed previous work on end-user debugging in different contexts, with the aim of extracting design guidelines for end-user debugging tools for trigger-action programming (Table 5.6). The analysis was guided by the following research questions:

RQ11) Which information, e.g., feedback and explanations, do end users need to understand, identify, and correct errors in IF-THEN rules?

RQ12) Which visual languages are more appropriate for debugging IF-THEN rules?

End-User Debugging: How to Avoid and Correct Errors (RQ11)

Debugging is the process of finding the cause of an identified misbehavior and fixing or removing it. Different previous studies, e.g., [104, 79], investigated how developers try to fix bugs, and discovered many slow, unproductive strategies. If it is challenging for programmers, the debugging process can become an insurmountable barrier for end users. In different contexts, ranging from spreadsheets [63]

Table 5.6: The design guidelines extracted by reviewing previous works on end-user debugging in different contexts.

Guideline	Description
GL1	A debugging tool for IF-THEN rules should empower users in frequently testing their solutions, e.g., by providing real-time feedback about possible run-time problems the rules may generate.
GL2	During the debugging of trigger-action rules it is important to provide users with tools for updating *on-the-fly* their solutions, e.g., to remove possible errors during the rule definition process.
GL3	In case of problems, a debugging tool for IF-THEN rules should provide users with textual and graphical explanations about the run-time behavior of the defined applications.
GL4	The Interrogative Debugging paradigm, with which users can ask questions like *"why something happens?"*, can be easily adapted to the event-driven nature of trigger-action rules.
GL5	Block programming based on the Jigsaw metaphor is understandable and easily adaptable to the definition of trigger-action rules.
GL6	The data-flow visual language is suitable for representing complex information such as the run-time behavior of a set of trigger-action rules.

to mashup programming [24], studies have demonstrated that end users try to fix problems by following a "debugging into existence" approach [122], i.e., they continuously twist and adapt their solutions until the failure "miraculously" goes away. Cao et al. [24], however, demonstrated that, if prompted with the right information, end users are also able to *design* applications and programs. In the context of mashup programming, for example, they proposed to add micro-evaluations of local portions of the mashup during the implementation phase, with the aim of reducing the effort of connecting the run-time output with the program's logic itself. We envision similar approaches also for our context, i.e., IF-THEN rules for personalizing IoT devices and online services. By providing real-time feedback during the definition of trigger-action rules, an EUD tool may empower users in frequently testing the correctness of their solutions (**GL1**), thus allowing them to update *on-the-fly* problematic rules (**GL2**), Table 5.6. This may increases the chances of fixing possible conflicts [24].

Previous studies on end-user debugging also highlight the benefits of providing users with textual and graphical explanations, to represent the run-time behaviors of the defined programs and their possible problems [88, 89] (**GL3**). Indeed, Ko et al. [77] discovered that programmers' questions at the time of failure are typically one of two types: *"why did"* questions, which assume the occurrence of an unexpected run-time action, and *"why didn't"* questions, which assume the absence of an expected run-time action [77]. The same authors extended the Alice programming

environment [138], a platform for creating interactive 3D virtual worlds, to support a "whyline" that allows users to receive answers concerning program outputs. Their work opened the way for a new paradigm, named *interrogative debugging*, that has been adopted in many different works on end-user debugging, ranging from tools to support more experienced developers [78] to interactive machine learning [81]. As preliminary suggested by Manca et al. [92], the interrogative debugging paradigm may effectively help end users debug their trigger-action rules (**GL4**). The event-driven nature of IF-THEN rules, in particular, naturally leads to questions such as *"why was this action executed?"* or *"why was this event not triggered?"*

Visual Languages for End-User Development (RQ12)

Besides the question of identifying which information end users need for debugging trigger-action rules, another important question is which visual languages are more appropriate in this context. Despite visual programming languages striving to simplify the intricate process of programming, in fact, they need to be tailored towards the domains in which they will be used [119]. The most common visual languages adopted in end-user development tools can be categorized into 3 main categories: a) form-filling, b) block programming, and c) data-flow.

Form-filling visual languages, also known as wizard-based languages, are extensively used in commercial platforms such as IFTTT and Zapier [46]. Also EUDebug [44] and ITAD [92], i.e., the first two works that explore end-user debugging in the context of trigger-action rules, exploit wizard-based interfaces. To define applications with the form-filling approach, be they rules or other types of programs, the user makes use of menus and fields to be completed. Tools that exploit form-filling visual languages, in particular, guide the user through a predefined, bounded procedure, by reducing the user interaction in completing a series of forms step-by-step. Despite form-filling approaches have been proved to be intuitive and easy to use for simple use cases, their closed form can be perceived as restrictive [117, 116].

Another popular approach in end-user development is block programming. A popular example of the approach can be seen in Scratch [120], a block-based visual programming language targeted primarily at children. With block programming, users can connect blocks of different sizes and shapes by dragging and dropping them on a work area. Different from form-filling approaches, tools based on block programming are less restrictive, and stimulate the user creativity. One of the most appreciated ways of representing blocks, in particular, is the Jigsaw metaphor. Here, blocks are represented as puzzle pieces that can be combined on the go, thus decreasing the learning curve and motivating users to explore the underlying tool. An application example is Puzzle, a visual environment for opportunistically creating mobile applications in mobile phones [42]. We envision that block programming approaches based on the Jigsaw metaphor could be easily adapted to the definition of IF-THEN rules for IoT personalization (**GL5**).

Finally, the last category of visual languages commonly adopted in EUD is data-flow. Different from the previous approaches, which were useful for simple use case such as the definition of a single rule, the process-oriented nature of data-flow programming languages makes them one of the best choice to represent complex use cases [16]. Process-oriented notations have been employed to provide increased expressiveness while still retaining easy-to-comprehend visualizations [121, 41]. The expressiveness of such notations, however, is often coupled with complex user interfaces [16]. This makes them difficult to be used at definition time, but useful to visualize complex information such as triggers, actions, and their relationships. For this reason, we envision that a data-flow visual language could be adopted for representing the behavior of multiple trigger-action rules (**GL6**), with the aim of helping users understand and identify unwanted run-time behaviors.

5.3.3 The *My IoT Puzzle* System

We integrated the extracted guidelines (Table 5.6) in *My IoT Puzzle*, a novel tool for defining and debugging IF-THEN rules. Under the hood, the tool exploits the SCPN RESTful server (Section 5.1.4), thus allowing the definition and the debug of IFTTT rules through the identification of loops, inconsistencies, and redundancies. The user interface of *My IoT Puzzle* has been implemented with the the Angular framework, by exploiting the jQuery[5] and Bootstrap[6] libraries. The interface iteratively assists end users in defining and debugging IF-THEN rules in 3 main phases, i.e., *definition*, *problem detection*, and *problem resolution*.

Definition

IF-THEN rules are defined through a block programming approach based on the Jigsaw metaphor (**GL5**). To design the definition metaphor 3 researchers produced and evaluated different mockups.

Figure 5.9 shows an example of the mockups produced: to avoid complex solutions, we decided to use 2 types of puzzle pieces, only, one for triggers and one for actions. Triggers and actions are therefore represented as complementary puzzle pieces that can be dragged and dropped in a Drop Area.

Figure 5.10 shows an example of the *definition* phase. A user selects a device (her *Android* smartphone) on which monitoring an event, and drops a specific trigger on the drop area (*"You enter an area"*). Then, she completes the trigger details by specifying the geographical area of her home. The tool uses initial feedback to preliminary allow the user assess her solution (**GL1**). Indeed, due to the complementary nature of the puzzle pieces, some wrong operations are prevented by

[5] https://jquery.com/, last visited on November 12, 2019

[6] https://getbootstrap.com/, last visited on November 12, 2019

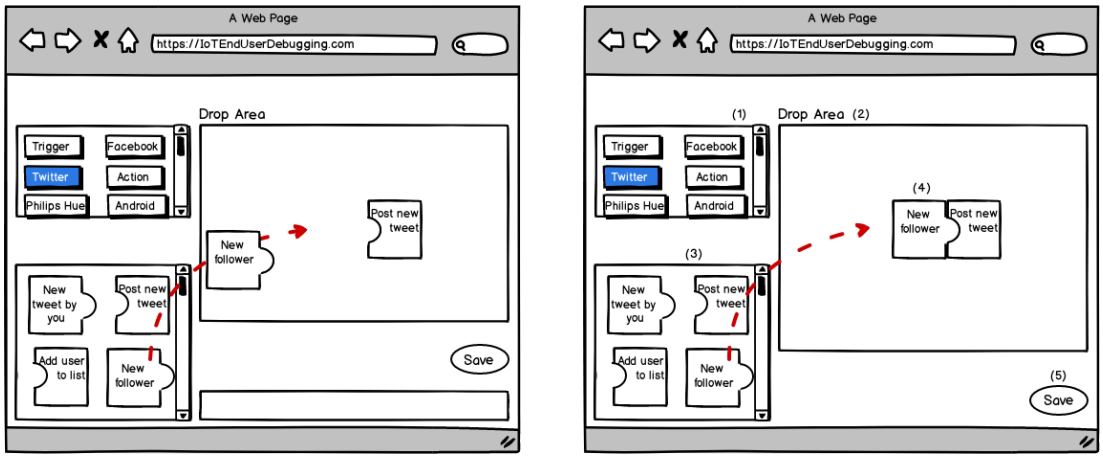


Figure 5.9: Two mockups produced to design the definition metaphor

construction: pieces of the same type, e.g., two trigger pieces, cannot be connected. Furthermore, as shown in Figure 5.10, the dropped trigger piece is worn, since it has been already used in other rules. In *My IoT Puzzle*, in particular, puzzle pieces deteriorate over time according to their usage history. Using the same trigger in multiple rules, in fact, means that the involved rules will be executed at the same time, thus increasing the chances of introducing conflicts such as redundancies and inconsistencies.

Problem Detection

The *problem detection* phase starts every time that *My IoT Puzzle* detects loops, inconsistencies, or redundancies during the *definition* phase. Figure 5.11 shows an example of the *problem detection* phase. The user selects her kitchen *Philips Hue* lamp, and she connects to the *"You enter an area"* trigger an action (*"Turn off lights"*) that is inconsistent with some previously saved rules. Therefore, the system warns the user with a red feedback (**GL1**), and allows her to get more information on how to solve the issue.

Problem Resolution

The *problem resolution* phase helps the user understand and fix the inconsistency conflicts detected during the previous phases (Figure 5.12). The user can see textual and graphical explanations of the detected problem (**GL3**). For graphically explaining the inconsistency, a data-flow visual language is used (**GL6**). Instead, the textual explanation follows the Interrogative Debugging paradigm (**GL4**), by explicitly describing *why* the problem is happening. Both the textual and graphical

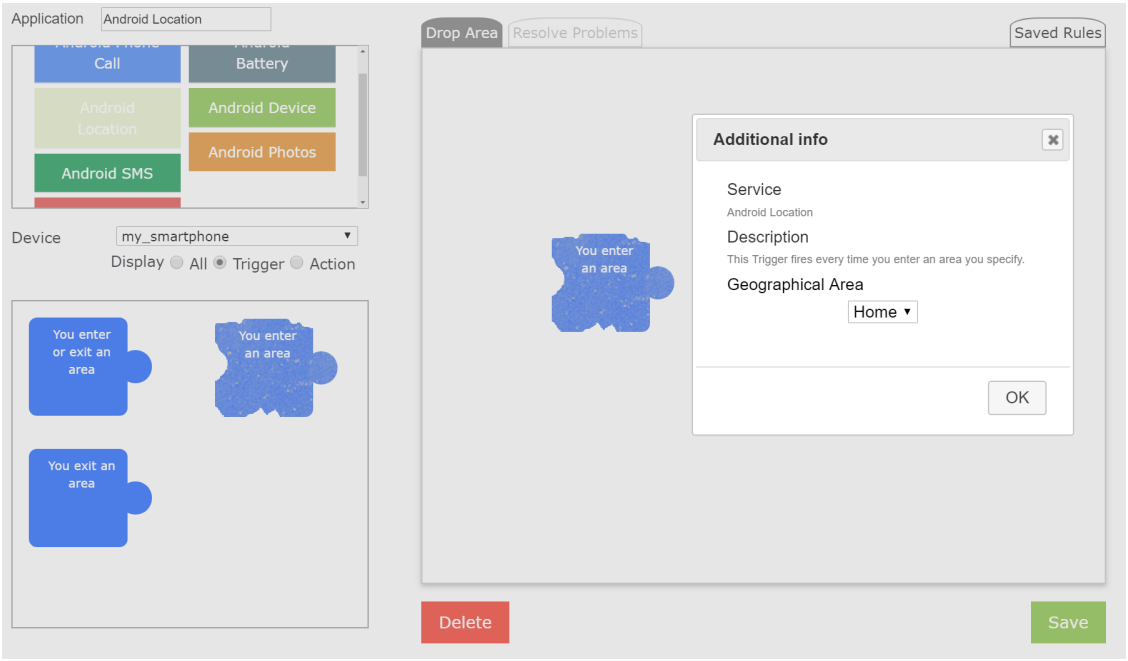


Figure 5.10: The definition of a new IF-THEN rule starts with the drag & drop of a new trigger on the Drop Area. The tool provides the user with an initial feedback: the piece of puzzle is worn, since it has been already used in other rules of the same user.

explanations, in particular, show that there is a saved rule that shares the same trigger, i.e., entering the home geographical area, but with an inconsistent action, i.e., turning on the kitchen *Philips Hue* lamp. The user has the possibility of updating on-the-fly the problematic rule by changing the trigger, the action, and/or the related details (**GL2**).

5.3.4 User Evaluation

We preliminary evaluated *My IoT Puzzle* through an exploratory study with 6 participants. Our aim was to assess the implemented design guidelines. The following question, in particular, guided our study:

RQ13) Do the different features offered by *My IoT Puzzle*, ranging from the provided feedback to the graphical and textual explanations, help participants correctly understand and fix potential conflicts in IF-THEN rules?

Participants

We recruited 6 university students (3 males and 3 females) with a mean age of 21.5 years (SD = 2.88) who had very limited or no experience in computer science

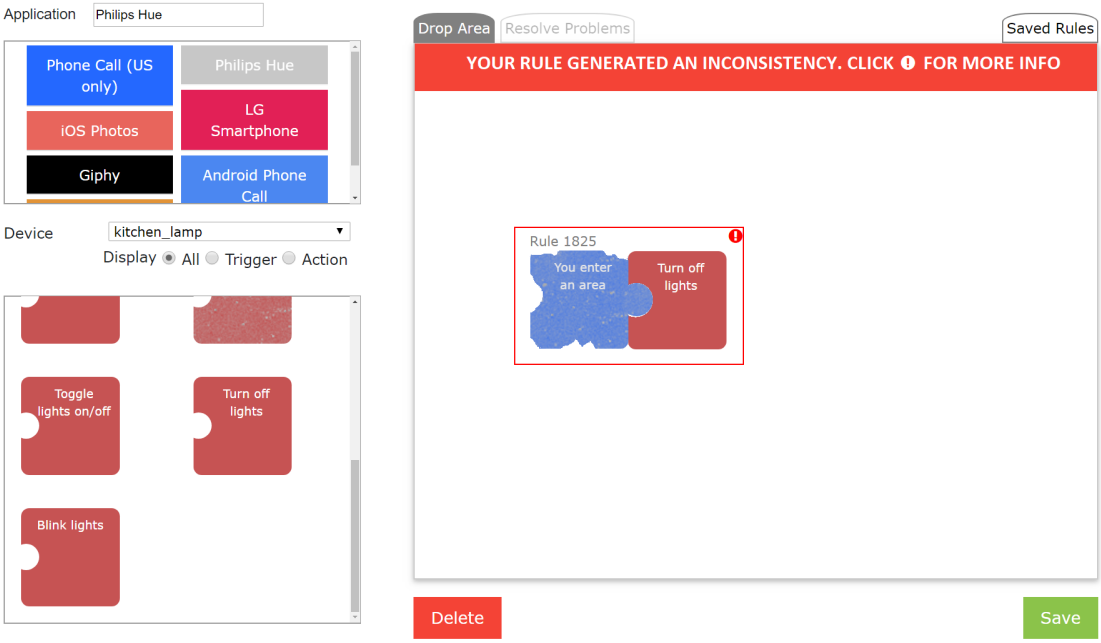


Figure 5.11: The user connects to the trigger an action that is inconsistent with some previously saved rules. The system warns the user with a red feedback.

and programming: on a Likert scale from 1 (No knowledge at all) to 5 (Expert), participants declared their experience with programming (M = 1.16, SD = 0.40) and with the trigger-action approach (M = 1.00, SD = 0).

Procedure

We brought each participant to our lab for a 45-minute session using *My IoT Puzzle*. At the beginning of the study, participants were introduced to trigger-action programming and the evaluated tool with an example of a rule definition. We then presented a task requiring the definition of 12 rules (*definition* phase). We reused, in particular, the same rules exploited in the *EUDebug* user study (Table 5.3). Rules were presented one at a time on a sheet of paper in a counterbalanced order, and were artificially constructed to generate 2 inconsistencies, 2 redundancies, and 1 loop. When *My IoT Puzzle* highlighted some problems (*problem detection* phase), participants were free to decide whether to save, update, or delete the problematic rule (*problem resolution* phase). All the sessions were video recorded for further analysis.

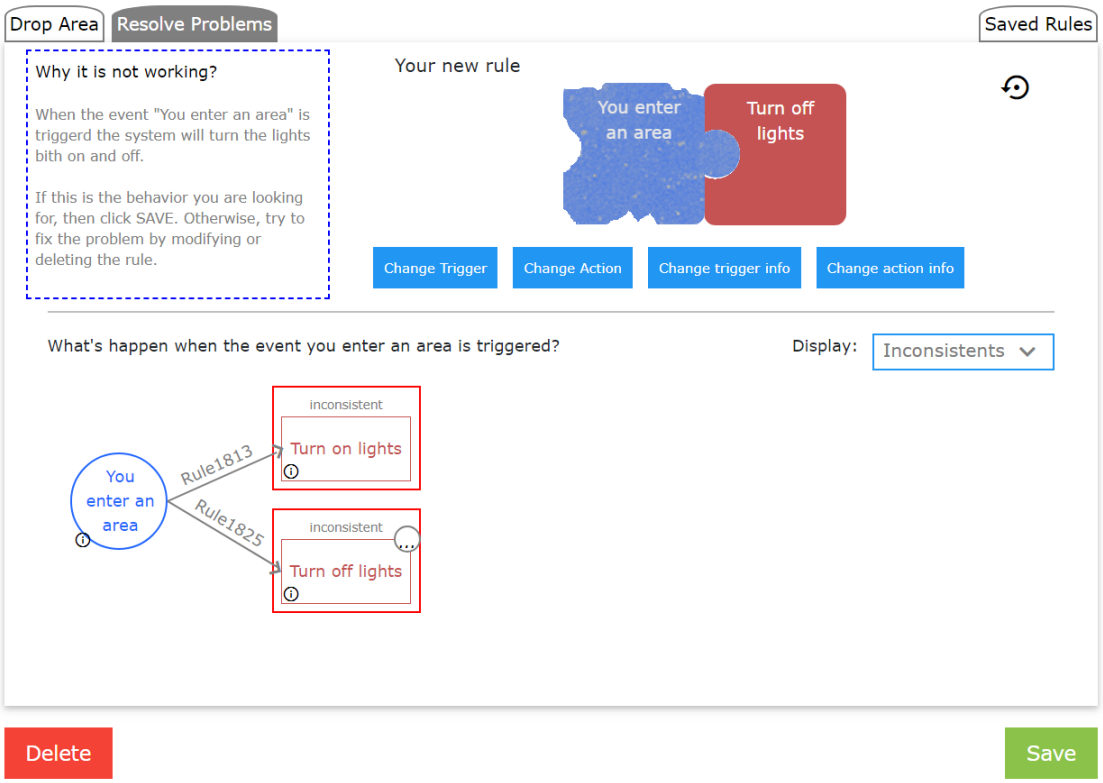


Figure 5.12: By opening the Resolve Problems area, the user can see textual and graphical explanation of the inconsistency, and she can resolve the problem by changing the rule.

Measures

We quantitatively measured the number of problematic rules that were **saved**, **updated**, or **deleted**. Furthermore, after each highlighted problem, we asked participants to qualitatively provide an **explanation** for their choices. When they decided to update or delete a rule that generated a problem, for example, they had to demonstrate to understand the problem by retrospectively explaining why the rule generated the issue. At the end of each session, we asked participants to quantitatively evaluate, on a Likert scale from 1 (Not understandable at all) to 5 (Very understandable), the **understandability** of a) the visual languages and feedback used in the *definition* and *problem detection* phases, b) the textual explanations, and c) the graphical explanations in the *problem resolution* phase. Finally, we performed a debriefing session with each participant.

5.3.5 Results

Table 5.7 reports the quantitative measures collected during the evaluation. In total, participants saved a rule that generated a problem in a limited number of cases, i.e., 3 out of 30 (10%), thus preliminary demonstrating that *My IoT Puzzle* helped them in identifying problems in trigger-action rules. In 2 cases, the saved rule generated a redundancy, while in the remaining cases a participant saved a rule that generated a loop. Participants deleted a rule that generated a problem in 18 cases out of 30 (60%), while they updated and successfully fixed the problem in 9 cases out of (30%). By analyzing the type of the problems, we found that rules that generated loops or redundancies were deleted most of the time (66.66% and 75%, respectively), while rules that generated an inconsistency were more frequently updated (58.33%) than deleted (41.67%). Such results are promising and suggest that feedback and explanations effectively assisted participants in understanding and fixing the highlighted problems (**RQ13**). Both loops and redundancies, indeed, result in some functionality that are replicated, thus motivating the deletion. Inconsistencies, instead, are typically caused by a mistake over a set of rules with different and specific purposes, thus making the "update" choice the most appropriate.

	Rules	**Deleted**	**Updated**	**Saved**
Loop	6	4 (66.66%)	1 (16.67%)	1 (16.67%)
Redundancy	12	9 (75%)	1 (8.33%)	2 (16.67%)
Inconsistency	12	5 (41.67%)	7 (58.33%)	0 (0%)
TOTAL	30	18 (60%)	9 (30%)	3 (10%)

Table 5.7: The number of times a rule that generated a problem was deleted, updated, or saved in the *My IoT Puzzle* study

We further investigated the results by analyzing the qualitative explanations given by the participants in case of a detected problem. We first tried to understand whether the participants who saved a problematic rule were aware of what would happened in the real world, or whether they simply made a mistake, thus unconsciously introducing a potential conflict at run-time. Both the users that saved a rule that generated a redundancy provided a sound explanation. When 2 rules simultaneously turned on the same lamp with different colors, for example, P1 said *"I don't care about the color, the important thing is that the lamp is turned on."* On the contrary, P3 failed in providing an explanation for saving the rule that generated a loop. She said *"I don't know how to solve the problem. I would save the rule, and then I would try the involved rules in the real world, to see what happens."* For what concerns the problematic rules that were deleted, participants provided a sound explanation in 17 cases out of 18 (94.44%). Only in 1 case a participant

discarded a rule without providing any explanation. Finally, participants made a reasonable change in all the rules that were updated, by successfully fixing the problem and by providing a sound explanation.

The promising results arising from the interaction between participants and *My IoT Puzzle* are confirmed by the answers they provided at the end of the study. Participants positively evaluated the understandability of the *definition* and *problem detection* phases (M = 4.50, SD = 0.54), and the understandability of the textual (M = 4.50, SD = 0.81) and graphical (M = 4.50, SD = 0.30) explanations in the *problem resolution* phase. Finally, users provided interesting suggestions to improve *My IoT Puzzle*. P1, for example, focused on the *definition* phase, by suggesting the possibility of defining multiple rules at the same time. P4 and P5, instead, focused on the *problem resolution* phase, and they asked to introduce recommendations and suggestions for updating problematic rules. P4, in particular, said that suggestions such as *"try to replace the trigger X with the trigger Y"* would allow non-expert users to better understand and fix the problem.

5.4 Discussion and Guidance for Future Research

In this chapter, we explored the urgent need of allowing end users to debug IF-THEN rules [46, 21, 14]. Unfortunately, even if providing end users with validation features and warning mechanisms could facilitate the adoption of EUD solutions in the real world [46], relatively little work has been done in this area. The complexity brought by the adoption of highly technology-dependent representation models, however, makes users often misinterpret the behavior of triggers and actions [17]. Users may therefore introduce errors in their IF-THEN rules that can potentially lead to unpredictable and even dangerous behaviors (*run-time problems* issue), e.g., a door that is unexpectedly unlocked.

To mitigate this issue, we started by formally characterizing control-flow problems that may arise in sets of IF-THEN rules, i.e., loops, inconsistencies, and redundancies, and we defined a novel formalism based on Petri Nets and the and the *EUPont* model and check IF-THEN rules at run-time. Then, we designed and implemented two different end-user debugging tools for trigger-action programming, namely *EUDdebug* and *My IoT Puzzle*.

EUDebug. With *EUDebug*, we added debugging features on top of an IFTTT-like interface. *EUDebug* highlights possible loops, inconsistencies, and redundancies that the defined rules may generate at run-time and allows their step-by-step simulation. We tested the tool in an exploratory study with 15 participants with the aim of preliminary investigating whether users without programming experience would be able to understand and debug IF-THEN

rules at definition time. Results were promising: with the help of *EUDebug*, participants successfully faced computer-related concepts such as loops, inconsistencies, and redundancies. Moreover, they were able to understand why their rules might generate a specific problem in most of the cases. Results also highlighted different perceptions among the various problems, i.e., participants demonstrated to be more tolerant with redundancies than with loops and inconsistencies. Furthermore, we found that highlighting a detected conflict, only, without providing any additional information, was often not sufficient for participants to understand the problem. Indeed, the step-by-step simulation was often a fundamental element for a successful debugging process, especially in case of complex problems like loops.

My IoT Puzzle. Stemming from these preliminary findings, we explored more specific questions about which visual languages and what information users would need to successfully debug IF-THEN rules. To this end, we extracted a set of guidelines for designing user interfaces for trigger-action programming from the literature, and we implemented them in *My IoT Puzzle*, a tool to define and debug IF-THEN rules based on the Jigsaw metaphor. The tool allows end users to define IF-THEN rules by representing triggers and actions as complementary puzzle pieces. Furthermore, it empowers end users to debug their rules through different real-time feedback, textual and graphical explanations, by following established theories such as the interrogative debugging paradigm [77]. Results of a preliminary user study were promising, and demonstrated that *My IoT Puzzle* was helpful for correctly identifying and fixing potential problems in trigger-action rules. Results also confirmed that the extracted design guidelines are valuable: the provided information and the exploited visual languages effectively helped participants understand, identify, and correct errors in trigger-action rules. The Jigsaw metaphor, for example, was appreciated by the participants, and turned to be easy to use and understand: all the participants defined the proposed IF-THEN rules without any problem. Also the feedback used in the *definition* phase turned to be useful for preliminary assessing the correctness of IF-THEN rules. When using a worn piece of puzzle, for example, P3 said *"now I'm going to make a mistake, I need to stay focused."* Furthermore, if the typical reaction to an highlighted problem was a mix of surprise and uncertainty, the *problem resolution* phase progressively made participants aware of the detected conflict: in most of the cases, the provided feedback and explanations allowed them to successfully fix the problem, either by deleting or updating the rule that generated it. This confirms that the usage of different representations of the same information facilitates users in analyzing problems [131].

Different questions and possible extensions can guide future research in this field. A first step to take is to move beyond control-flow problems, i.e., loops,

inconsistencies, and redundancies. The work of Brackenbury et al. [14] provides a comprehensive overview on ten types of programming bugs that might arise in IF-THEN rules. Besides control-flow problems, other important bugs influencing users' ability to predict the outcomes of IF-THEN rules are timing issues and inaccurate user expectations. Timing bugs relate to the difficulty of understanding when triggers and actions are actually executed, while inaccurate user expectations can include, for example, priority conflicts and secure-default biases, i.e., assuming that the system has a default state that is safe [148]. To assist users in dealing with these types of problems, different solutions could be adopted, ranging from warning mechanisms to simulated animations that graphically exemplify when and how the defined rules are potentially executed. Future works would also need to explore the application of the presented approaches and methodologies to trigger-action programming paradigms that include trigger conditions and multiple actions. Indeed, while the exploited *SCPN* formalism can be easily generalized, users may experience further difficulties when dealing with more complex rules.

Chapter 6

Conclusions

6.1 Summary of Contributions

In the current Internet of Things era, users are surrounded by a multitude of smart devices, always connected to the Internet, that can communicate with each other, with humans, and with the environment. Some of such devices, e.g., smartphones and PCs, serve as a gateway for a variety of online services such as social networks and messaging applications. The result is a complex (and growing) network of connected entities, either physical or virtual, that presents both opportunities and problems.

To take advantage of such a network, users must be able to personalize it. In this thesis, we presented a set of research works that aim to assist end users in easily and efficiently personalizing the functionality of their connected entities. We focused on one of the most common paradigms that allow end users to define custom applications over their own connected entities, i.e., trigger-action programming. Through rigorous user studies and controlled experiments, in particular, we reported on different approaches and practical solutions to improve the definition, the discovery, and the debugging of IF-THEN rules.

Summarizing, the main outcomes presented in this thesis are the following.

EUPont is an ontological high-level representation for end-user development that allows the definition of abstract and technology-independent IF-THEN rules that can be adapted to different contextual situations, independently of manufacturers, brands, and other technical details. The aim is to simplify the processes needed by end users to define personalizations: by defining IF-THEN rules such as *"if I enter a closed space, then cool the environment"*, users are not requested to specify technological details, and they can personalize the functionality of their connected entities with fewer rules, fewer mistakes, and in less time.

EUDoptimizer is an optimization tool that dynamically redesign layouts in trigger-action programming interfaces in an interactive way, i.e., by considering the choices made by end users during the rule definition process. The aim is to promote the discovery of the "right" connected entity to be used for defining the trigger or the action, according to the current user need: the tool moves towards the top of the interface layout the set of devices or online services that are typically used for defining triggers, or that are typically associated as an action to a defined trigger, while maintaining logical groups of semantically correlated items. The tool is based on *SDP-FSM*, a predictive model for trigger-action programming that exploits *EUPont* and the Search-Decision-Pointing model of human performance in menu search. Through *EUDoptimizer*, end users define IF-THEN rules in less time, since the tool reduces the cognitive load required to discover appropriate connected entities to be personalized.

RecRules is a hybrid and semantic recommendation system of IF-THEN rules. Its aim is to allow users to discover new rules on the basis of the underlying functionality, rather than the involved brands or manufacturers. A rule for turning on a *Philips Hue* lamp, for example, is *functionally* similar to a rule for opening the *Hunter Douglas* blinds, because they share a common final goal, i.e., to light up a place. *RecRules* outperforms state-of-the-art ranking-oriented and semantic recommendation algorithms. Its main characteristic, i.e., *recommending by functionality*, increases diversity and serendipity in the recommended rules while maintaining a high recommendation accuracy.

EUDebug is an end-user debugging tool built on top of an IFTTT-like interface that enables end users to debug their IF-THEN rules at composition time. It assists users in identifying rule conflicts, and it allows them to foresee the run-time behavior of their rules through step-by-step simulation. To model and check the run-time behavior of IF-THEN rules, we defined a novel formalism, named *SCPN*, based on Petri Nets and the *EUPont* model. With the help of *EUDebug*, users can successfully face computer-related concepts such as loops, inconsistencies, and redundancies. The step-by-step simulation, in particular, helps users understand why their rules might generate a specific problem.

My IoT Puzzle is an end-user debugging tool to compose and debug IF-THEN rules based on the Jigsaw metaphor. As *EUDebug*, it exploits the *SCPN* formalism, and it is based on a set of design guidelines extracted from the literature. *My IoT Puzzle* represents triggers and actions as complementary puzzle pieces, and it provides users with different real-time feedback, textual and graphical explanations, by following established theories such as the interrogative debugging paradigm. The usage of different representations and

visual languages facilitates users in analyzing problems and helps them understand, identify, and correct errors in IF-THEN rules.

6.2 Future Works

As reported at the end of the main chapters of this thesis, our work provides researchers with different insights to inspire further research in the fields of Internet of Things, end-user development, and trigger-action programming.

The usage of *EUPont* to define IF-THEN rules (Chapter 3), for instance, raises questions on the actual execution of abstract behaviors. Furthermore, moving towards a higher level of abstraction requires taking into account trustfulness, security, and privacy issues. If we imagine IF-THEN rules such as "if I enter a closed space, then cool the environment", which environments and connected entities is the user authorized to control? How can end users be authenticated for using public and shared IoT devices and services, and how can we taking into account the user privacy?

Also the approaches and tools we presented to assist users in discovering rules and related functionality (Chapter 4), i.e., *EUDoptimizer* and *RecRules* could open up new possibilities to design novel trigger-action programming platforms. Optimization methods and recommendation techniques, in particular, could be used to improve existing trigger-action programming platforms without requiring major changes in the underlying representation models. The same applies to end-user debugging tools, e.g., the ones we presented in Chapter 5. By extending the presented approaches and methodologies, debugging mechanisms may be also helpful to transitioning to more complex versions of the trigger-action programming paradigm, e.g., with trigger conditions and multiple actions.

Finally, all the presented works have been evaluated through "in-lab" user studies, mainly. Future works would need to test the proposed solutions for defining, discovering, and debugging IF-THEN rules "in the field," with real devices and online services. By observing users personalizing their own connected entities during their daily activities, researchers could better evaluate the efficacy of tools such as *EUDebug* and *RecRules*, along with their acceptance.

Appendix A

Publications

April: January, 2020.

A.1 International Journals

1. Fulvio Corno, Luigi De Russis, Alberto Monge Roffarello (2019) **RecRules: Recommending IF-THEN Rules for End-User Development** in: ACM Transactions on Intelligent Systems and Technology, ACM, pages: 27, Volume 10
   ISSN: 2157-6904
   DOI: 10.1145/3344211

2. Fulvio Corno, Luigi De Russis, Alberto Monge Roffarello (2019) **EUDoptimizer: Assisting End Users in Composing IF-THEN Rules Through Optimization** in: IEEE Access, IEEE, pages: 11, Volume 7
   ISSN: 2169-3536
   DOI: 10.1109/ACCESS.2019.2905619

3. Fulvio Corno, Luigi De Russis, Alberto Monge Roffarello (2019) **A High-Level Semantic Approach to End-User Development in the Internet of Things** in: International Journal of Human-Computer Studies, Elsevier, pages: 14, Volume 125
   ISSN: 1071-5819
   DOI: 10.1016/j.ijhcs.2018.12.008

4. Fulvio Corno, Luigi De Russis, Alberto Monge Roffarello (2018) **AwareNotifications: Multi-Device Semantic Notification Handling with User-Defined Preferences** in: Journal of Ambient Intelligence and Smart Environments, IOS Press, pages: 17, Volume 10
   ISSN: 1876-1364
   DOI: 10.3233/AIS-180492

5. Fulvio Corno, Luigi De Russis, Alberto Monge Roffarello (2017) **A Semantic Web Approach to Simplifying Trigger-Action Programming in the IoT** in: IEEE Computer, IEEE, pages: 7, Volume 50,
ISSN: 0018-9162, DOI: 10.1109/MC.2017.4041355

A.2 Proceedings

1. Fulvio Corno, Luigi De Russis, Alberto Monge Roffarello (*in press*) **TAPrec: Supporting the Composition of Trigger-Action Rules Through Dynamic Recommendations** in: IUI '20: ACM International Conference on Intelligent User Interfaces, ACM, Cagliari (IT), March 17-20, 2020, pages: 10
DOI: 10.1145/3377325.3377499

2. Alberto Monge Roffarello, Luigi De Russis (2019) **Towards Detecting and Mitigating Smartphone Habits** in: Proceedings of the 2019 ACM International Joint Conference and 2019 International Symposium on Pervasive and Ubiquitous Computing and Wearable Computers (UbiComp 2019), ACM, London (UK), September 10-14, 2019
DOI: 10.1145/3341162.3343770]

3. Fulvio Corno, Luigi De Russis, Alberto Monge Roffarello (2019) **My IoT Puzzle: Debugging IF-THEN Rules Through the Jigsaw Metaphor** in: IS-EUD: the 7th International Symposium on End-User Development, Springer, Hertfordshire (UK), July 10-12, 2019, pages: 16
DOI: 10.1007/978-3-030-24781-2_2

4. Fulvio Corno, Luigi De Russis, Alberto Monge Roffarello (2019) **Empowering End Users in Debugging Trigger-Action Rules** in: Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems (CHI '19), ACM, Glasgow (UK), May 4-9, 2019, pages: 13
DOI: 10.1145/3290605.3300618

5. Alberto Monge Roffarello, Luigi De Russis (2019) **The Race Towards Digital Wellbeing: Issues and Opportunities** in: Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems (CHI '19), ACM, Glasgow (UK), May 4-9, 2019, pages: 14
DOI: 10.1145/3290605.3300616

6. Alberto Monge Roffarello (2018) **End User Development in the IoT: a Semantic Approach** in: Proceedings of 14th International Conference on Intelligent Environments, IEEE, Rome (Italy), June 25-28, 2018, pages: 4
DOI: 10.1109/IE.2018.00026

7. Luigi De Russis, Alberto Monge Roffarello (2018) **A Debugging Approach for Trigger-Action Programming** in: Proceedings of the 2018 CHI Conference Extended Abstracts on Human Factors in Computing Systems, ACM, Montreal, QC (Canada), April 21–26, 2018, pages: 6
DOI: 10.1145/3170427.3188641

8. Luigi De Russis, Alberto Monge Roffarello (2017) **On the Benefit of Adding User Preferences to Notification Delivery** in: Proceedings of the 2017 CHI Conference Extended Abstracts on Human Factors in Computing Systems, ACM, Denver, CO (USA), May 6–11, 2017, pages: 7
DOI: 10.1145/3027063.3053160

9. Fulvio Corno, Luigi De Russis, Alberto Monge Roffarello (2017) **A High-Level Approach Towards End User Development in the IoT** in: Proceedings of the 2017 CHI Conference Extended Abstracts on Human Factors in Computing Systems, ACM, Denver, CO (USA), May 6–11, 2017, pages: 7
DOI: 10.1145/3027063.3053157

10. Fulvio Corno, Luigi De Russis, Alberto Monge Roffarello (2016) **A Healthcare Support System for Assisted Living Facilities: an IoT Solution** in: 2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC), IEEE, Atlanta, Georgia (USA), June 10-14, 2016, pages: 9
DOI: 10.1109/COMPSAC.2016.29

A.3 Book Chapters

1. Fulvio Corno, Luigi De Russis, Alberto Monge Roffarello (2018) **IoT for Ambient Assisted Living: Care4Me - A Healthcare Support System** in: Wearable Technologies: Concepts, Methodologies, Tools, and Applications (reprint), IGI Global, pages: 27, Chapter 9
ISBN: 9781522554851

2. Fulvio Corno, Luigi De Russis, Alberto Monge Roffarello (2017) **IoT for Ambient Assisted Living: Care4Me - A Healthcare Support System** in: Internet of Things and Advanced Application in Healthcare, pages: 32, Chapter 3
ISBN: 1522518207

Bibliography

[1] Pierre A. Akiki, Arosha K. Bandara, and Yijun Yu. "Visual Simple Transformations: Empowering End-Users to Wire Internet of Things Objects". In: *ACM Transactions on Computer-Human Interaction* 24.2 (Apr. 2017), 10:1–10:43. ISSN: 1073-0516. DOI: 10.1145/3057857.

[2] Sarabjot Singh Anand, Patricia Kearney, and Mary Shapcott. "Generating Semantically Enriched User Profiles for Web Personalization". In: *ACM Transactions on Internet Technology* 7.4 (Oct. 2007). ISSN: 1533-5399. DOI: 10.1145/1278366.1278371.

[3] Carmelo Ardito, Paolo Buono, Giuseppe Desolda, and Maristella Matera. "From smart objects to smart experiences: An end-user development approach". In: *International Journal of Human-Computer Studies* 114 (2018). Advanced User Interfaces for Cultural Heritage, pp. 51–68. ISSN: 1071-5819. DOI: https://doi.org/10.1016/j.ijhcs.2017.12.002.

[4] Luigi Atzori, Antonio Iera, and Giacomo Morabito. "The Internet of Things: A Survey". In: *Computer Networks: The International Journal of Computer and Telecommunications Networking* 54 (Oct. 2010), pp. 2787–2805. ISSN: 1389-1286. DOI: 10.1016/j.comnet.2010.05.010.

[5] Juan Carlos Augusto and Miguel J. Hornos. "Software simulation and verification to increase the reliability of Intelligent Environments". In: *Advances in Engineering Software* 58.Supplement C (2013), pp. 18–34. ISSN: 0965-9978. DOI: 10.1016/j.advengsoft.2012.12.004.

[6] James Bailey, Guozhu Dong, and Kotagiri Ramamohanarao. "On the decidability of the termination problem of active database systems". In: *Theoretical Computer Science* 311.1 (2004), pp. 389–437. ISSN: 0304-3975. DOI: 10.1016/j.tcs.2003.09.003.

[7] Gilles Bailly, Antti Oulasvirta, Timo Kötzing, and Sabrina Hoppe. "MenuOptimizer: Interactive Optimization of Menu Systems". In: *Proceedings of the 26th Annual ACM Symposium on User Interface Software and Technology*. UIST '13. St. Andrews, Scotland, United Kingdom: ACM, 2013, pp. 331–342. ISBN: 978-1-4503-2268-3. DOI: 10.1145/2501988.2502024.

[8] Payam Barnaghi, Wei Wang, Cory Henson, and Kerry Taylor. "Semantics for the Internet of Things: Early Progress and Back to the Future". In: *International Journal on Semantic Web and Information Systems* 8.1 (Jan. 2012), pp. 1–21. ISSN: 1552-6283. DOI: 10.4018/jswis.2012010101.

[9] Barbara Rita Barricelli and Stefano Valtolina. "End-User Development: 5th International Symposium, IS-EUD 2015, Madrid, Spain, May 26-29, 2015. Proceedings". In: Cham, Germany: Springer International Publishing, 2015. Chap. Designing for End-User Development in the Internet of Things, pp. 9–24. ISBN: 978-3-319-18425-8. DOI: 10.1007/978-3-319-18425-8_2.

[10] Maria Bermudez-Edo, Tarek Elsaleh, Payam Barnaghi, and Kerry Taylor. "IoT-Lite: A Lightweight Semantic Model for the Internet of Things". In: *Proceedings of 13th International Conference on Ubiquitous Intelligence and Computing*. (in press). 2016.

[11] Christian Bizer, Tom Heath, and Tim Berners-Lee. "Linked data - the story so far". In: *International Journal on Semantic Web and Information System* 5.3 (2009), pp. 1–22.

[12] Dario Bonino and Luigi De Russis. "DogOnt as a viable seed for semantic modeling of AEC/FM". In: *Semantic Web* 9.6 (2018), pp. 763–780. ISSN: 1570-0844. DOI: 10.3233/SW-180295.

[13] Mike Botts, George Percivall, Carl Reed, and John Davidson. "OGC Sensor Web Enablement: Overview and High Level Architecture". In: *GeoSensor Networks*. Ed. by Silvia Nittel, Alexandros Labrinidis, and Anthony Stefanidis. Springer-Verlag, 2008, pp. 175–190. ISBN: 978-3-540-79995-5. DOI: 10.1007/978-3-540-79996-2_10.

[14] Will Brackenbury, Abhimanyu Deora, Jillian Ritchey, Jason Vallee, Weijia He, Guan Wang, Michael L. Littman, and Blase Ur. "How Users Interpret Bugs in Trigger-Action Programming". In: *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. CHI '19. Glasgow, Scotland Uk: ACM, 2019, 552:1–552:12. ISBN: 978-1-4503-5970-2. DOI: 10.1145/3290605.3300782.

[15] Leo Breiman. "Random Forests". In: *Machine Learning* 45.1 (Oct. 2001), pp. 5–32. ISSN: 0885-6125. DOI: 10.1023/A:1010933404324.

[16] Julia Brich, Marcel Walch, Michael Rietzler, Michael Weber, and Florian Schaub. "Exploring End User Programming Needs in Home Automation". In: *ACM Transaction on Computer-Human Interaction* 24.2 (Apr. 2017), 11:1–11:35. ISSN: 1073-0516. DOI: 10.1145/3057858.

[17] A.J. Bernheim Brush, Bongshin Lee, Ratul Mahajan, Sharad Agarwal, Stefan Saroiu, and Colin Dixon. "Home Automation in the Wild: Challenges and Opportunities". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '11. Vancouver, BC, Canada: ACM, 2011, pp. 2115–2124. ISBN: 978-1-4503-0228-9. DOI: 10.1145/1978942.1979249.

[18] Margaret Burnett, Curtis Cook, Omkar Pendse, Gregg Rothermel, Jay Summet, and Chris Wallace. "End-user Software Engineering with Assertions in the Spreadsheet Paradigm". In: *Proceedings of the 25th International Conference on Software Engineering*. ICSE '03. Portland, Oregon: IEEE Computer Society, 2003, pp. 93–103. ISBN: 0-7695-1877-X.

[19] Raymond J. Butler. "Is this spreadsheet a tax evader? How HM Customs and Excise test spreadsheet applications". In: *Proceedings of the 33rd Annual Hawaii International Conference on System Sciences*. Jan. 2000, 6 pp. vol.1-. DOI: 10.1109/HICSS.2000.926737.

[20] Michael D. Byrne, John R. Anderson, Scott Douglass, and Michael Matessa. "Eye Tracking the Visual Search of Click-down Menus". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '99. Pittsburgh, Pennsylvania, USA: Association for Computing Machinery, 1999, pp. 402–409. ISBN: 0201485591. DOI: 10.1145/302979.303118. URL: https://doi.org/10.1145/302979.303118.

[21] Danilo Caivano, Daniela Fogli, Rosa Lanzilotti, Antonio Piccinno, and Fabio Cassano. "Supporting end users to control their smart home: design implications from a literature review and an empirical investigation". In: *Journal of Systems and Software* 144 (2018), pp. 295–313. ISSN: 0164-1212. DOI: https://doi.org/10.1016/j.jss.2018.06.035.

[22] Julio Cano, Gwenaël Delaval, and Eric Rutten. "Coordination of ECA Rules by Verification and Control". In: *Coordination Models and Languages*. Ed. by Eva Kühn and Rosario Pugliese. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 33–48. ISBN: 978-3-662-43376-8.

[23] Iván Cantador, Alejandro Bellogin, and Pablo Castells. "A Multilayer Ontology-based Hybrid Recommendation Model". In: *AI Communications* 21.2-3 (Apr. 2008), pp. 203–210. ISSN: 0921-7126.

[24] Jill Cao, Kyle Rector, Thomas H. Park, Scott D. Fleming, Margaret M. Burnett, and Susan Wiedenbeck. "A Debugging Perspective on End-User Mashup Programming". In: *2010 IEEE Symposium on Visual Languages and Human-Centric Computing*. Sept. 2010, pp. 149–156. DOI: 10.1109/VLHCC.2010.29.

[25] Vint Cerf and Max Senges. "Taking the Internet to the Next Physical Level". In: *IEEE Computer* 49.2 (Feb. 2016), pp. 80–86. ISSN: 0018-9162. DOI: 10.1109/MC.2016.51.

[26] Andy Cockburn, Carl Gutwin, and Saul Greenberg. "A Predictive Model of Menu Performance". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '07. San Jose, California, USA: ACM, 2007, pp. 627–636. ISBN: 978-1-59593-593-9. DOI: 10.1145/1240624.1240723.

[27] Michael Compton, Payam Barnaghi, Luis Bermudez, Raúl García-Castro, Oscar Corcho, Simon Cox, John Graybeal, Manfred Hauswirth, Cory Henson, Arthur Herzog, Vincent Huang, Krzysztof Janowicz, W. David Kelsey, Danh Le Phuoc, Laurent Lefort, Myriam Leggieri, Holger Neuhaus, Andriy Nikolov, Kevin Page, Alexandre Passant, Amit Sheth, and Kerry Taylor. "The SSN ontology of the W3C semantic sensor network incubator group". In: *Web Semantics: Science, Services and Agents on the World Wide Web* 17 (2012), pp. 25–32. ISSN: 1570-8268. DOI: 10.1016/j.websem.2012.05.003.

[28] David T. Connolly. "An improved annealing scheme for the QAP". In: *European Journal of Operational Research* 46.1 (1990), pp. 93–100. ISSN: 0377-2217. DOI: 10.1016/0377-2217(90)90301-Q.

[29] Diane Cook and Sajal Das. *Smart Environments: Technology, Protocols and Applications*. Vol. 43. Wiley Series on Parallel and Distributed Computing. Wiley-Interscience, 2004. ISBN: 0471544485.

[30] Fulvio Corno, Luigi De Russis, and Alberto Monge Roffarello. "A High-Level Approach Towards End User Development in the IoT". In: *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems*. CHI EA '17. Denver, Colorado, USA: ACM, 2017, pp. 1546–1552. ISBN: 978-1-4503-4656-6. DOI: 10.1145/3027063.3053157.

[31] Fulvio Corno, Luigi De Russis, and Alberto Monge Roffarello. "A high-level semantic approach to End-User Development in the Internet of Things". In: *International Journal of Human-Computer Studies* 125 (2019), pp. 41–54. ISSN: 1071-5819. DOI: 10.1016/j.ijhcs.2018.12.008.

[32] Fulvio Corno, Luigi De Russis, and Alberto Monge Roffarello. "A Semantic Web Approach to Simplifying Trigger-Action Programming in the IoT". In: *Computer* 50.11 (Nov. 2017), pp. 18–24. ISSN: 0018-9162. DOI: 10.1109/MC.2017.4041355.

[33] Fulvio Corno, 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*. CHI '19. Glasgow, Scotland Uk: Association for Computing Machinery, 2019. ISBN: 9781450359702. DOI: 10.1145/3290605.3300618.

[34] Fulvio Corno, Luigi De Russis, and Alberto Monge Roffarello. "EUDoptimizer: Assisting End Users in Composing IF-THEN Rules Through Optimization". In: *IEEE Access* 7 (2019), pp. 37950–37960. DOI: 10.1109/ACCESS.2019.2905619.

[35] Fulvio Corno, Luigi De Russis, and Alberto Monge Roffarello. "My IoT Puzzle: Debugging IF-THEN Rules Through the Jigsaw Metaphor". In: *End-User Development*. Cham: Springer International Publishing, 2019, pp. 18–33. ISBN: 978-3-030-24781-2.

[36] Fulvio Corno, Luigi De Russis, and Alberto Monge Roffarello. "RecRules: Recommending IF-THEN Rules for End-User Development". In: *ACM Transactions on Intelligent Systems and Technology* 10.5 (Sept. 2019), 58:1–58:27. ISSN: 2157-6904. DOI: 10.1145/3344211.

[37] Fulvio Corno, Luigi De Russis, and Alberto Monge Roffarello. "TAPrec: Supporting the Composition of Trigger-Action Rules through Dynamic Recommendations". In: *Proceedings of the 25th International Conference on Intelligent User Interfaces*. IUI '20. Cagliari, Italy: Association for Computing Machinery, 2020, pp. 579–588. ISBN: 9781450371186. DOI: 10.1145/3377325.3377499.

[38] Irena Pletikosa Cvijikj and Florian Michahelles. "Architecting the Internet of Things". In: ed. by Dieter Uckelmann, Mark Harrison, and Florian Michahelles. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011. Chap. The Toolkit Approach for End-user Participation in the Internet of Things, pp. 65–96. ISBN: 978-3-642-19157-2. DOI: 10.1007/978-3-642-19157-2_4.

[39] Allen Cypher. "EAGER: Programming Repetitive Tasks by Example". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '91. New Orleans, Louisiana, USA: ACM, 1991, pp. 33–39. ISBN: 0-89791-383-3. DOI: 10.1145/108844.108850.

[40] Andrea Renika D'Souza, Di Yang, and Cristina V. Lopes. "Collective Intelligence for Smarter API Recommendations in Python". In: *2016 IEEE 16th International Working Conference on Source Code Analysis and Manipulation (SCAM)*. Oct. 2016, pp. 51–60. DOI: 10.1109/SCAM.2016.22.

[41] Yngve Dahl and Reidar-Martin Svendsen. "End-User Composition Interfaces for Smart Environments: A Preliminary Study of Usability Factors". In: *Design, User Experience, and Usability. Theory, Methods, Tools and Practice*. Ed. by Aaron Marcus. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 118–127. ISBN: 978-3-642-21708-1.

[42] Jose Danado and Fabio Paternò. "Puzzle: A mobile application development environment using a jigsaw metaphor". In: *Journal of Visual Languages & Computing* 25.4 (2014), pp. 297–315. ISSN: 1045-926X. DOI: http://dx.doi.org/10.1016/j.jvlc.2014.03.005.

[43] Florian Daniel and Maristella Matera. *Mashups: Concepts, Models and Architectures*. Springer Publishing Company, Incorporated, 2014. ISBN: 3642550487, 9783642550485.

[44] Luigi De Russis and Alberto Monge Roffarello. "A Debugging Approach for Trigger-Action Programming". In: *Extended Abstracts of the 2018 CHI Conference on Human Factors in Computing Systems*. CHI EA '18. Montreal QC, Canada: Association for Computing Machinery, 2018. ISBN: 9781450356213. DOI: 10.1145/3170427.3188641.

[45] Suparna De, Tarek Elsaleh, Payam Barnaghi, and Stefan Meissner. "An Internet of Things Platform for Real-World and Digital Objects". In: *Scalable Computing: Practice and Experience* 13.1 (2012), pp. 45–57. URL: http://epubs.surrey.ac.uk/531903/.

[46] Giuseppe Desolda, Carmelo Ardito, and Maristella Matera. "Empowering End Users to Customize Their Smart Environments: Model, Composition Paradigms, and Domain-Specific Tools". In: *ACM Transactions on Computer-Human Interaction* 24.2 (2017), 12:1–12:52. ISSN: 1073-0516. DOI: 10.1145/3057859.

[47] Anind K. Dey, Timothy Sohn, Sara Streng, and Justin Kodama. "iCAP: Interactive Prototyping of Context-aware Applications". In: *Proceedings of the 4th International Conference on Pervasive Computing*. PERVASIVE'06. Dublin, Ireland: Springer-Verlag, 2006, pp. 254–271. DOI: 10.1007/11748625_16.

[48] Marco Dorigo and Luca Maria Gambardella. "Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem". In: *Trans. Evol. Comp* 1.1 (Apr. 1997), pp. 53–66. ISSN: 1089-778X. DOI: 10.1109/4235.585892.

[49] Ekwa Duala-Ekoko and Martin P. Robillard. "Using Structure-Based Recommendations to Facilitate Discoverability in APIs". In: *ECOOP 2011 – Object-Oriented Programming: 25th European Conference, Lancaster, Uk, July 25-29, 2011 Proceedings*. Ed. by Mira Mezini. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 79–104. ISBN: 978-3-642-22655-7. DOI: 10.1007/978-3-642-22655-7_5.

[50] Natalya F. Noy and Deborah L. Mcguinness. *Ontology Development 101: A Guide to Creating Your First Ontology*. Tech. rep. Stanford University, 2001.

[51] Ignacio Fernández-Tobias, Iván Cantador, Marius Kaminskas, and Francesco Ricci. "A Generic Semantic-based Framework for Cross-domain Recommendation". In: *Proceedings of the 2Nd International Workshop on Information Heterogeneity and Fusion in Recommender Systems*. HetRec '11. Chicago,

Illinois: ACM, 2011, pp. 25–32. ISBN: 978-1-4503-1027-7. DOI: 10.1145/2039320.2039324.

[52] Gerhard Fischer. "End-User Development and Meta-design: Foundations for Cultures of Participation". In: *End-User Development*. Ed. by Volkmar Pipek, Mary Beth Rosson, Boris de Ruyter, and Volker Wulf. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 3–14. ISBN: 978-3-642-00427-8.

[53] Yoav Freund, Raj Iyer, Robert E. Schapire, and Yoram Singer. "An Efficient Boosting Algorithm for Combining Preferences". In: *Journal of Machine Learning Research* 4 (Dec. 2003), pp. 933–969. ISSN: 1532-4435.

[54] Jerome Friedman, Trevor Hastie, and Rob Tibshirani. "Regularization Paths for Generalized Linear Models via Coordinate Descent". In: *Journal of Statistical Software* 33.1 (2010), pp. 1–22. URL: http://www.jstatsoft.org/v33/i01/.

[55] Krzysztof Z. Gajos, Jacob O. Wobbrock, and Daniel S. Weld. "Improving the Performance of Motor-impaired Users with Automatically-generated, Ability-based Interfaces". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '08. Florence, Italy: ACM, 2008, pp. 1257–1266. ISBN: 978-1-60558-011-1. DOI: 10.1145/1357054.1357250.

[56] Zeno Gantner, Lucas Drumond, Christoph Freudenthaler, Steffen Rendle, and Lars Schmidt-Thieme. "Learning Attribute-to-Feature Mappings for Cold-Start Recommendations". In: *Proceedings of the 2010 IEEE International Conference on Data Mining*. ICDM '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 176–185. ISBN: 978-0-7695-4256-0. DOI: 10.1109/ICDM.2010.129.

[57] Zeno Gantner, Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. "MyMediaLite: A Free Recommender System Library". In: *Proceedings of the Fifth ACM Conference on Recommender Systems*. RecSys '11. Chicago, Illinois, USA: ACM, 2011, pp. 305–308. ISBN: 978-1-4503-0683-6. DOI: 10.1145/2043932.2043989.

[58] Andrés Garcia-silva, Oscar Corcho, Harith Alani, and Asunción Gómez-pérez. "Review: Review of the State of the Art: Discovering and Associating Semantics to Tags in Folksonomies". In: *The Knowledge Engineering Review* 27.1 (Feb. 2012), pp. 57–85. ISSN: 0269-8889. DOI: 10.1017/S026988891100018X.

[59] Stella Gatzio Gatziu and K. R. Dittrich. "Detecting composite events in active database systems using Petri nets". In: *Proceedings of IEEE International Workshop on Research Issues in Data Engineering: Active Databases Systems*. Feb. 1994, pp. 2–9. DOI: 10.1109/RIDE.1994.282859.

[60] Giuseppe Ghiani, Marco Manca, Fabio Paternò, and Carmen Santoro. "Personalization of Context-Dependent Applications Through Trigger-Action Rules". In: *ACM Transactions on Computer-Human Interaction* 24.2 (2017), 14:1–14:33. ISSN: 1073-0516. DOI: 10.1145/3057861.

[61] Birte Glimm, Ian Horrocks, Boris Motik, Giorgos Stoilos, and Zhe Wang. "HermiT: An OWL 2 Reasoner". In: *Journal of Automated Reasoning* 53.3 (Oct. 2014), pp. 245–269. ISSN: 0168-7433. DOI: 10.1007/s10817-014-9305-1.

[62] Fred Glover and Manuel Laguna. *Tabu Search*. Norwell, MA, USA: Kluwer Academic Publishers, 1997. ISBN: 079239965X.

[63] Valentina Grigoreanu, Margaret Burnett, Susan Wiedenbeck, Jill Cao, Kyle Rector, and Irwin Kwan. "End-user Debugging Strategies: A Sensemaking Perspective". In: *ACM Transaction on Computer-Human Interaction* 19.1 (May 2012), 5:1–5:28. ISSN: 1073-0516. DOI: 10.1145/2147783.2147788.

[64] Will Haines, Melinda Gervasio, Aaron Spaulding, and Bart Peintner. "Recommendations for End-User Development". In: *Proceedings of the ACM RecSys 2010 Workshop on User-Centric Evaluation of Recommender Systems and Their Interfaces (UCERSTI)*. 2010.

[65] Mostafa Hamza and Robert J. Walker. "Recommending Features and Feature Relationships from Requirements Documents for Software Product Lines". In: *Proceedings of the Fourth International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering*. RAISE '15. Florence, Italy: IEEE Press, 2015, pp. 25–31.

[66] Benjamin Heitmann and Conor Hayes. "Using linked data to build open, collaborative recommender systems". In: *Artificial Intelligence* (2010), pp. 76–81.

[67] Matthew Horridge and Sean Bechhofer. "The OWL API: A Java API for OWL Ontologies". In: *Semantic Web* 2.1 (Jan. 2011), pp. 11–21. ISSN: 1570-0844. DOI: 10.3233/SW-2011-0025.

[68] Matthew Horridge and Sean Bechhofer. "The OWL API: A Java API for OWL Ontologies". In: *Semantic Web* 2.1 (Jan. 2011), pp. 11–21. ISSN: 1570-0844.

[69] Peizhao Hu, Jadwiga Indulska, and Ricky Robinson. "An Autonomic Context Management System for Pervasive Computing". In: *2008 Sixth Annual IEEE International Conference on Pervasive Computing and Communications (PerCom)*. Mar. 2008, pp. 213–223. DOI: 10.1109/PERCOM.2008.56.

[70] Justing Huang and Maya Cakmak. "Supporting Mental Model Accuracy in Trigger-action Programming". In: *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*. UbiComp '15. Osaka, Japan: ACM, 2015, pp. 215–225. ISBN: 978-1-4503-3574-4. DOI: 10.1145/2750858.2805830.

[71] Ting-Hao K. Huang, A. Azaria, and J. P. Bigham. "InstructableCrowd: Creating IF-THEN Rules via Conversations with the Crowd". In: *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems*. CHI EA '16. Santa Clara, California, USA: ACM, 2016, pp. 1555–1562. ISBN: 978-1-4503-4082-3. DOI: 10.1145/2851581.2892502.

[72] *IFTTT*. Accessed: 2019-11-20. 2019. URL: https://ifttt.com/.

[73] Kurt Jensen. *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use, Vol. 2*. London, UK, UK: Springer-Verlag, 1995. ISBN: 3-540-58276-2.

[74] Xiaoqing Jin, Yousra Lembachar, and Gianfranco Ciardo. "Symbolic Termination and Confluence Checking for ECA Rules". In: *Transactions on Petri Nets and Other Models of Concurrency IX*. Ed. by Maciej Koutny, Serge Haddad, and Alex Yakovlev. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 99–123. ISBN: 978-3-662-45730-6. DOI: 10.1007/978-3-662-45730-6_6.

[75] Houda Khrouf and Raphaël Troncy. "Hybrid Event Recommendation Using Linked Data and User Diversity". In: *Proceedings of the 7th ACM Conference on Recommender Systems*. RecSys '13. Hong Kong, China: ACM, 2013, pp. 185–192. ISBN: 978-1-4503-2409-0. DOI: 10.1145/2507157.2507171.

[76] Cory Kissinger, Margaret Burnett, Simone Stumpf, Neeraja Subrahmaniyan, Laura Beckwith, Sherry Yang, and Mary Beth Rosson. "Supporting End-user Debugging: What Do Users Want to Know?" In: *Proceedings of the Working Conference on Advanced Visual Interfaces*. AVI '06. Venezia, Italy: ACM, 2006, pp. 135–142. ISBN: 1-59593-353-0. DOI: 10.1145/1133265.1133293.

[77] Andrew J. Ko and Brad A. Myers. "Designing the Whyline: A Debugging Interface for Asking Questions About Program Behavior". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '04. Vienna, Austria: ACM, 2004, pp. 151–158. ISBN: 1-58113-702-8. DOI: 10.1145/985692.985712.

[78] Andrew J. Ko and Brad A. Myers. "Finding Causes of Program Output with the Java Whyline". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '09. Boston, MA, USA: ACM, 2009, pp. 1569–1578. ISBN: 978-1-60558-246-7. DOI: 10.1145/1518701.1518942.

[79] Andrew J. Ko, Brad A. Myers, Michael L. Coblenz, and Htet Htet Aung. "An Exploratory Study of How Developers Seek, Relate, and Collect Relevant Information during Software Maintenance Tasks". In: *IEEE Transactions on Software Engineering* 32.12 (Dec. 2006), pp. 971–987. ISSN: 0098-5589. DOI: 10.1109/TSE.2006.116.

[80] Jacob Krüger. "When to Extract Features: Towards a Recommender System". In: *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*. ICSE '18. Gothenburg, Sweden: ACM, 2018, pp. 518–520. ISBN: 978-1-4503-5663-3. DOI: 10.1145/3183440.3190328.

[81] Todd Kulesza, 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*. IUI '15. Atlanta, Georgia, USA: ACM, 2015, pp. 126–137. ISBN: 978-1-4503-3306-1. DOI: 10.1145/2678025.2701399.

[82] Ni Lao and William W. Cohen. "Relational Retrieval Using a Combination of Path-constrained Random Walks". In: *Mach. Learn.* 81.1 (Oct. 2010), pp. 53–67. ISSN: 0885-6125. DOI: 10.1007/s10994-010-5205-8.

[83] Eric Lee and James Macgregor. "Minimizing User Search Time in Menu Retrieval Systems". In: *Human Factors* 27.2 (1985), pp. 157–162. DOI: 10.1177/001872088502700203.

[84] Lukas Lerche and Dietmar Jannach. "Using Graded Implicit Feedback for Bayesian Personalized Ranking". In: *Proceedings of the 8th ACM Conference on Recommender Systems*. RecSys '14. Foster City, Silicon Valley, California, USA: ACM, 2014, pp. 353–356. ISBN: 978-1-4503-2668-1. DOI: 10.1145/2645710.2645759.

[85] Xiaoou Li, Joselito Medina, and Sergio Chapa. "Applying Petri Nets in Active Database Systems". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 37.4 (July 2007), pp. 482–493. ISSN: 1094-6977. DOI: 10.1109/TSMCC.2007.897329.

[86] Chieh-Jan Mike Liang, Lei Bu, Zhao Li, Junbei Zhang, Shi Han, Börje F. Karlsson, Dongmei Zhang, and Feng Zhao. "Systematically Debugging IoT Control System Correctness for Building Automation". In: *Proceedings of the 3rd ACM International Conference on Systems for Energy-Efficient Built Environments*. BuildSys '16. Palo Alto, CA, USA: Association for Computing Machinery, 2016, pp. 133–142. ISBN: 9781450342643. DOI: 10.1145/2993422.2993426.

[87] Henry Lieberman, Fabio Paternò, Markus Klann, and Volker Wulf. "End User Development". In: ed. by Henry Lieberman, Fabio Paternò, and Volker Wulf. Dordrecht, Netherlands: Springer Netherlands, 2006. Chap. End-User Development: An Emerging Paradigm, pp. 1–8. ISBN: 978-1-4020-5386-3. DOI: 10.1007/1-4020-5386-X_1.

[88] Brian Y. Lim and Anind K. Dey. "Toolkit to Support Intelligibility in Context-aware Applications". In: *Proceedings of the 12th ACM International Conference on Ubiquitous Computing*. UbiComp '10. Copenhagen, Denmark: ACM, 2010, pp. 13–22. ISBN: 978-1-60558-843-8. DOI: 10.1145/1864349.1864353.

[89] Brian Y. Lim, Anind K. Dey, and Daniel Avrahami. "Why and Why Not Explanations Improve the Intelligibility of Context-aware Intelligent Systems". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '09. Boston, MA, USA: ACM, 2009, pp. 2119–2128. ISBN: 978-1-60558-246-7. DOI: 10.1145/1518701.1519023.

[90] Tie-Yan Liu. "Learning to Rank for Information Retrieval". In: *Foundations and Trends In Information Retrieval* 3.3 (Mar. 2009), pp. 225–331. ISSN: 1554-0669. DOI: 10.1561/1500000016.

[91] Yuri Malheiros, Alan Moraes, Cleyton Trindade, and Silvio Meira. "A Source Code Recommender System to Support Newcomers". In: *2012 IEEE 36th Annual Computer Software and Applications Conference*. July 2012, pp. 19–24. DOI: 10.1109/COMPSAC.2012.11.

[92] Marco Manca, Fabio, Paternò, Carmen Santoro, and Luca Corcella. "Supporting end-user debugging of trigger-action rules for IoT applications". In: *International Journal of Human-Computer Studies* 123 (2019), pp. 56–69. ISSN: 1071-5819.

[93] Toshiyuki Masui and Ken Nakayama. "Repeat and Predict: Two Keys to Efficient Text Editing". In: *Conference Companion on Human Factors in Computing Systems*. CHI '94. Boston, Massachusetts, USA: ACM, 1994, pp. 31–32. ISBN: 0-89791-651-4. DOI: 10.1145/259963.259999.

[94] Shouichi Matsui and Seiji Yamada. "Genetic Algorithm Can Optimize Hierarchical Menus". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '08. Florence, Italy: ACM, 2008, pp. 1385–1388. ISBN: 978-1-60558-011-1. DOI: 10.1145/1357054.1357271.

[95] Friedemann Mattern and Christian Floerkemeier. "From the Internet of Computers to the Internet of Things". In: *From Active Data Management to Event-Based Systems and More: Papers in Honor of Alejandro Buchmann on the Occasion of His 60th Birthday*. Ed. by Kai Sachs, Ilia Petrov, and Pablo Guerrero. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 242–259. ISBN: 978-3-642-17226-7. DOI: 10.1007/978-3-642-17226-7_15.

[96] Simon Mayer, Andreas Tschofen, Anind K. Dey, and Friedemann Mattern. "User Interfaces for Smart Things – A Generative Approach with Semantic Interaction Descriptions". In: *ACM Transaction on Computer-Human Interaction* 21.2 (Feb. 2014). ISSN: 1073-0516. DOI: 10.1145/2584670.

[97] Sean M. McNee, John Riedl, and Joseph A. Konstan. "Being Accurate is Not Enough: How Accuracy Metrics Have Hurt Recommender Systems". In: *CHI '06 Extended Abstracts on Human Factors in Computing Systems*. CHI EA '06. Montreal QC, Canada: ACM, 2006, pp. 1097–1101. ISBN: 1-59593-298-4. DOI: 10.1145/1125451.1125659.

[98] Kim Mens and Angela Lozano. "Source Code-Based Recommendation Systems". In: *Recommendation Systems in Software Engineering*. Ed. by Martin P. Robillard, Walid Maalej, Robert J. Walker, and Thomas Zimmermann. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 93–130. ISBN: 978-3-642-45135-5. DOI: 10.1007/978-3-642-45135-5_5.

[99] Xianghang Mi, Feng Qian, Ying Zhang, and XiaoFeng Wang. "An Empirical Characterization of IFTTT: Ecosystem, Usage, and Performance". In: *Proceedings of the 2017 Internet Measurement Conference*. IMC '17. London, United Kingdom: ACM, 2017, pp. 398–404. ISBN: 978-1-4503-5118-8. DOI: 10.1145/3131365.3131369.

[100] *Microsoft Flow*. Accessed: 2019-11-20. 2019. URL: https://flow.microsoft.com/en-us/.

[101] Bamshad Mobasher, Xin Jin, and Yanzan Zhou. "Web Mining: From Web to Semantic Web". In: *Web Mining: From Web to Semantic Web: First European Web Mining Forum, EWMF 2003, Cavtat-Dubrovnik, Croatia, September 22, 2003, Invited and Selected Revised Papers*. Ed. by Bettina Berendt, Andreas Hotho, Dunja Mladenič, Maarten van Someren, Myra Spiliopoulou, and Gerd Stumme. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004. Chap. Semantically Enhanced Collaborative Filtering on the Web, pp. 57–76. ISBN: 978-3-540-30123-3. DOI: 10.1007/978-3-540-30123-3_4.

[102] Alberto Monge Roffarello. "End User Development in the IoT: A Semantic Approach". In: *2018 14th International Conference on Intelligent Environments (IE)*. June 2018, pp. 107–110. DOI: 10.1109/IE.2018.00026.

[103] Dejan Munjin. "User Empowerment in the Internet of Things". eng. PhD thesis. Université de Genève, May 2013. URL: http://archive-ouverte.unige.ch/unige:28951.

[104] Brad A. Myers, Andrew J. Ko, Chris Scaffidi, Stephen Oney, YoungSeok Yoon, Kerry Chang, Mary Beth Kery, and Toby Jia-Jun Li. "Making End

User Development More Natural". In: *New Perspectives in End-User Development*. Ed. by Fabio Paternò and Volker Wulf. Cham: Springer International Publishing, 2017, pp. 1–22. ISBN: 978-3-319-60291-2. DOI: 10.1007/978-3-319-60291-2_1.

[105] Abdallah Namoun, Athanasia Daskalopoulou, Nikolay Mehandjiev, and Zhang Xun. "Exploring Mobile End User Development: Existing Use and Design Factors". In: *IEEE Transactions on Software Engineering* 42.10 (Oct. 2016), pp. 960–976. ISSN: 0098-5589. DOI: 10.1109/TSE.2016.2532873.

[106] Anh Tuan Nguyen, Michael Hilton, Mihai Codoban, Hoan Anh Nguyen, Lily Mast, Eli Rademacher, Tien N. Nguyen, and Danny Dig. "API Code Recommendation Using Statistical Learning from Fine-grained Changes". In: *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. FSE 2016. Seattle, WA, USA: ACM, 2016, pp. 511–522. ISBN: 978-1-4503-4218-6. DOI: 10.1145/2950290.2950333.

[107] Xia Ning and George Karypis. "SLIM: Sparse Linear Methods for Top-N Recommender Systems". In: *2011 IEEE 11th International Conference on Data Mining*. Dec. 2011, pp. 497–506. DOI: 10.1109/ICDM.2011.134.

[108] Xia Ning and George Karypis. "Sparse Linear Methods with Side Information for Top-n Recommendations". In: *Proceedings of the Sixth ACM Conference on Recommender Systems*. RecSys '12. Dublin, Ireland: ACM, 2012, pp. 155–162. ISBN: 978-1-4503-1270-7. DOI: 10.1145/2365952.2365983.

[109] Tommaso Di Noia, Vito Claudio Ostuni, Paolo Tomeo, and Eugenio Di Sciascio. "SPrank: Semantic Path-Based Ranking for Top-N Recommendations Using Linked Open Data". In: *ACM Transactions on Intelligent Systems and Technology* 8.1 (Sept. 2016), 9:1–9:34. ISSN: 2157-6904. DOI: 10.1145/2899005.

[110] Sergio Oramas, Vito Claudio Ostuni, Tommaso Di Noia, Xavier Serra, and Eugenio Di Sciascio. "Sound and Music Recommendation with Knowledge Graphs". In: *ACM Trans. Intell. Syst. Technol.* 8.2 (Oct. 2016), 21:1–21:21. ISSN: 2157-6904. DOI: 10.1145/2926718. URL: http://doi.acm.org/10.1145/2926718.

[111] Vito Claudio Ostuni, Tommaso Di Noia, Eugenio Di Sciascio, and Roberto Mirizzi. "Top-N Recommendations from Implicit Feedback Leveraging Linked Open Data". In: *Proceedings of the 7th ACM Conference on Recommender Systems*. RecSys '13. Hong Kong, China: ACM, 2013, pp. 85–92. ISBN: 978-1-4503-2409-0. DOI: 10.1145/2507157.2507172.

[112] Antti Oulasvirta. "User Interface Design with Combinatorial Optimization". In: *Computer* 50.1 (Jan. 2017), pp. 40–47. ISSN: 0018-9162. DOI: 10.1109/MC.2017.6.

[113] Antti Oulasvirta, Anna Reichel, Wenbin Li, Yan Zhang, Myroslav Bachynskyi, Keith Vertanen, and Per Ola Kristensson. "Improving Two-thumb Text Entry on Touchscreen Devices". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '13. Paris, France: ACM, 2013, pp. 2765–2774. ISBN: 978-1-4503-1899-0. DOI: 10.1145/2470654.2481383.

[114] Raymond R. Panko. "What We Know About Spreadsheet Errors". In: *J. End User Comput.* 10.2 (May 1998), pp. 15–21. ISSN: 1063-2239. URL: http://dl.acm.org/citation.cfm?id=287893.287899.

[115] Singiresu S. Rao. *Engineering Optimization: Theory and Practice: Fourth Edition*. John Wiley and Sons, June 2009. ISBN: 9780470183526. DOI: 10.1002/9780470549124.

[116] Michaela R. Reisinger, Johann Schrammel, and Peter Fröhlich. "Visual end-user programming in smart homes: Complexity and performance". In: *2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. Oct. 2017, pp. 331–332.

[117] Michaela R. Reisinger, Johann Schrammel, and Peter Fröhlich. "Visual languages for smart spaces: End-user programming between data-flow and form-filling". In: *2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. Oct. 2017, pp. 165–169.

[118] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. "BPR: Bayesian Personalized Ranking from Implicit Feedback". In: *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*. UAI '09. Montreal, Quebec, Canada: AUAI Press, 2009, pp. 452–461. ISBN: 978-0-9749039-5-8.

[119] Alexander Repenning and Tamara Sumner. "Agentsheets: A Medium for Creating Domain-Oriented Visual Languages". In: *Computer* 28.3 (Mar. 1995), pp. 17–25. ISSN: 0018-9162.

[120] Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian Silverman, and Yasmin Kafai. "Scratch: Programming for All". In: *Commun. ACM* 52.11 (Nov. 2009), pp. 60–67. ISSN: 0001-0782.

[121] Michael Rietzler, Julia Greim, Marcel Walch, Florian Schaub, Björn Wiedersheim, and Michael Weber. "homeBLOX: Introducing Process-driven Home Automation". In: *Proceedings of the 2013 ACM Conference on Pervasive and Ubiquitous Computing Adjunct Publication*. UbiComp '13 Adjunct. Zurich, Switzerland: ACM, 2013, pp. 801–808. ISBN: 978-1-4503-2215-7.

[122] Jochen Rode and Mary Beth Rosson. "Programming at Runtime: Requirements and Paradigms for Nonprogrammer Web Application Development". In: *Proceedings of the 2003 IEEE Symposium on Human Centric Computing Languages and Environments*. HCC '03. Washington, DC, USA: IEEE Computer Society, 2003, pp. 23–30. ISBN: 0-7803-8225-0.

[123] Paulo Salem. "User Interface Optimization Using Genetic Programming with an Application to Landing Pages". In: *Proc. ACM Hum.-Comput. Interact.* 1.1 (June 2017), 13:1–13:17. ISSN: 2573-0142. DOI: 10.1145/3099583.

[124] Giovanni Semeraro, Pasquale Lops, Pierpaolo Basile, and Marco de Gemmis. "Knowledge Infusion into Content-based Recommender Systems". In: *Proceedings of the Third ACM Conference on Recommender Systems*. RecSys '09. New York, New York, USA: ACM, 2009, pp. 301–304. ISBN: 978-1-60558-435-5. DOI: 10.1145/1639714.1639773.

[125] Wei Shen, Jianyong Wang, and Jiawei Han. "Entity Linking with a Knowledge Base: Issues, Techniques, and Solutions". In: *IEEE Transactions on Knowledge and Data Engineering* 27.2 (Feb. 2015), pp. 443–460. ISSN: 1041-4347. DOI: 10.1109/TKDE.2014.2327028.

[126] Yue Shi, Alexandros Karatzoglou, Linas Baltrunas, Martha Larson, Alan Hanjalic, and Nuria Oliver. "TFMAP: Optimizing MAP for Top-n Context-aware Recommendation". In: *Proceedings of the 35th International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '12. Portland, Oregon, USA: ACM, 2012, pp. 155–164. ISBN: 978-1-4503-1472-5. DOI: 10.1145/2348283.2348308.

[127] *SmartRules*. Accessed: 2019-11-20. 2019. URL: http://smartrulesapp.com/.

[128] We are Social. *Digital in 2020*. https://wearesocial.com/blog/2020/01/digital-2020-3-8-billion-people-use-social-media. 2020.

[129] Harald Steck. "Evaluation of Recommendations: Rating-prediction and Ranking". In: *Proceedings of the 7th ACM Conference on Recommender Systems*. RecSys '13. Hong Kong, China: ACM, 2013, pp. 213–220. ISBN: 978-1-4503-2409-0. DOI: 10.1145/2507157.2507160.

[130] Kathryn T. Stolee and Sebastian Elbaum. "Identification, Impact, and Refactoring of Smells in Pipe-Like Web Mashups". In: *IEEE Transactions on Software Engineering* 39.12 (Dec. 2013), pp. 1654–1679. ISSN: 0098-5589. DOI: 10.1109/TSE.2013.42.

[131] Neeraja Subrahmaniyan, Cory Kissinger, Kyle Rector, Derek Inman, Jared Kaplan, Laura Beckwith, and Margaret Burnett. "Explaining Debugging Strategies to End-User Programmers". In: *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing*. VLHCC '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 127–136. ISBN: 0-7695-2987-9.

[132] Zhu Sun, Jie Yang, Jie Zhang, Alessandro Bozzon, Long-Kai Huang, and Chi Xu. "Recurrent Knowledge Graph Embedding for Effective Recommendation". In: *Proceedings of the 12th ACM Conference on Recommender Systems*. RecSys '18. Vancouver, British Columbia, Canada: ACM, 2018, pp. 297–305. ISBN: 978-1-4503-5901-6. DOI: 10.1145/3240323.3240361.

[133] Milijana Surbatovich, Jassim Aljuraidan, Lujo Bauer, Anupam Das, and Limin Jia. "Some Recipes Can Do More Than Spoil Your Appetite: Analyzing the Security and Privacy Risks of IFTTT Recipes". In: *Proceedings of the 26th International Conference on World Wide Web*. WWW '17. Perth, Australia: International World Wide Web Conferences Steering Committee, 2017, pp. 1501–1510. ISBN: 9781450349130. DOI: 10.1145/3038912.3052709.

[134] Kashyap Todi, Daryl Weir, and Antti Oulasvirta. "Sketchplore: Sketch and Explore with a Layout Optimiser". In: *Proceedings of the 2016 ACM Conference on Designing Interactive Systems*. DIS '16. Brisbane, QLD, Australia: ACM, 2016, pp. 543–555. ISBN: 978-1-4503-4031-1. DOI: 10.1145/2901790.2901817.

[135] Khai Truong, Elaine Huang, and Gregory Abowd. "UbiComp 2004: Ubiquitous Computing: 6th International Conference, Nottingham, UK, September 7-10, 2004. Proceedings". In: ed. by Nigel Davies, Elizabeth D. Mynatt, and Itiro Siio. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004. Chap. CAMP: A Magnetic Poetry Interface for End-User Programming of Capture Applications for the Home, pp. 143–160. ISBN: 978-3-540-30119-6. DOI: 10.1007/978-3-540-30119-6_9.

[136] Blase Ur, Elyse McManus, Melwyn Pak Yong Ho, and Michael L. Littman. "Practical Trigger-action Programming in the Smart Home". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '14. Toronto, Ontario, Canada: ACM, 2014, pp. 803–812. ISBN: 978-1-4503-2473-1. DOI: 10.1145/2556288.2557420. URL: http://doi.acm.org/10.1145/2556288.2557420.

[137] Blase Ur, Melwyn Pak Yong Ho, Stephen Brawner, Jiyun Lee, Sarah Mennicken, Noah Picard, Diane Schulze, and Michael L. Littman. "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. New York, NY, USA: ACM, 2016, pp. 3227–3231. DOI: 10.1145/2858036.2858556.

[138] UVa User Interface Group. "Alice: Rapid Prototyping for Virtual Reality". In: *IEEE Computer Graphics and Applications* 15.3 (May 1995), pp. 8–11. ISSN: 0272-1716. DOI: 10.1109/38.376600.

[139] Claudia Vannucchi, Michelangelo Diamanti, Gianmarco Mazzante, Diletta Cacciagrano, Rosario Culmone, Nikos Gorogiannis, Leonardo Mostarda, and Franco Raimondi. "Symbolic verification of event–condition–action rules in intelligent environments". In: *Journal of Reliable Intelligent Environments* 3.2 (Aug. 2017), pp. 117–130. ISSN: 2199-4676. DOI: 10.1007/s40860-017-0036-z.

[140] Wei Wang, Suparna De, Gilbert Cassar, and Klaus Moessner. "Knowledge Representation in the Internet of Things: Semantic Modelling and its Applications". In: *Automatika - Journal for Control, Measurement, Electronics, Computing and Communications* 54.4 (Oct. 2013), pp. 388–400. URL: http://epubs.surrey.ac.uk/794745/.

[141] *WebThings Gateway*. Accessed: 2019-11-20. 2019. URL: https://iot.mozilla.org/gateway/.

[142] Markus Weimer, Alexandros Karatzoglou, and Alex Smola. "Improving Maximum Margin Matrix Factorization". In: *Mach. Learn.* 72.3 (Sept. 2008), pp. 263–276. ISSN: 0885-6125. DOI: 10.1007/s10994-008-5073-7.

[143] Mickey R. Wilhelm and Thomas L. Ward. "Solving Quadratic Assignment Problems by Simulated Annealing". In: *IIE Transactions* 19.1 (1987), pp. 107–119. DOI: 10.1080/07408178708975376.

[144] Qiang Wu, Christopher J. Burges, Krysta M. Svore, and Jianfeng Gao. "Adapting Boosting for Information Retrieval Measures". In: *Information Retrieval Journal* 13.3 (June 2010), pp. 254–270. ISSN: 1386-4564. DOI: 10.1007/s10791-009-9112-1.

[145] Yao Wu, Christopher DuBois, Alice X. Zheng, and Martin Ester. "Collaborative Denoising Auto-Encoders for Top-N Recommender Systems". In: *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining*. WSDM '16. San Francisco, California, USA: ACM, 2016, pp. 153–162. ISBN: 978-1-4503-3716-8. DOI: 10.1145/2835776.2835837.

[146] Stephen Yang, Alex Lee, William Chu, and Hongji Yang. "Rule base verification using Petri nets". In: *Computer Software and Applications Conference, 1998. COMPSAC '98. Proceedings. The Twenty-Second Annual International*. Aug. 1998, pp. 476–481. DOI: 10.1109/CMPSAC.1998.716699.

[147] Lina Yao, Quan Z. Sheng, Anne H.H. Ngu, Helen Ashman, and Xue Li. "Exploring Recommendations in Internet of Things". In: *Proceedings of the 37th International ACM SIGIR Conference on Research & Development in Information Retrieval*. SIGIR '14. Gold Coast, Queensland, Australia: ACM, 2014, pp. 855–858. ISBN: 978-1-4503-2257-7. DOI: 10.1145/2600428.2609458.

[148] Svetlana Yarosh and Pamela Zave. "Locked or Not? Mental Models of IoT Feature Interaction". In: *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. CHI '17. Denver, Colorado, USA: Association for Computing Machinery, 2017, pp. 2993–2997. ISBN: 9781450346559. DOI: 10.1145/3025453.3025617.

[149] Yunwen Ye and Gerhard Fischer. "Supporting reuse by delivering task-relevant and personalized information". In: *Proceedings of the 24th International Conference on Software Engineering. ICSE 2002*. May 2002, pp. 513–523. DOI: 10.1109/ICSE.2002.1007995.

[150] Xiao Yu, Xiang Ren, Yizhou Sun, Quanquan Gu, Bradley Sturt, Urvashi Khandelwal, Brandon Norick, and Jiawei Han. "Personalized Entity Recommendation: A Heterogeneous Information Network Approach". In: *Proceedings of the 7th ACM International Conference on Web Search and Data Mining*. WSDM '14. New York, New York, USA: ACM, 2014, pp. 283–292. ISBN: 978-1-4503-2351-2. DOI: 10.1145/2556195.2556259.

[151] *Zapier*. Accessed: 2019-11-20. 2019. URL: https://zapier.com/.

[152] Arkady Zaslavsky and Prem Prakash Jayaraman. "Discovery in the Internet of Things: The Internet of Things (Ubiquity Symposium)". In: *Ubiquity* 2015.October (Oct. 2015), 2:1–2:10. ISSN: 1530-2180. DOI: 10.1145/2822529.

[153] Shumin Zhai, Michael Hunter, and Barton A. Smith. "The Metropolis Keyboard - an Exploration of Quantitative Techniques for Virtual Keyboard Design". In: *Proceedings of the 13th Annual ACM Symposium on User Interface Software and Technology*. UIST '00. San Diego, California, USA: ACM, 2000, pp. 119–128. ISBN: 1-58113-212-3. DOI: 10.1145/354401.354424.

[154] Lefan Zhang, Weijia He, Jesse Martinez, Noah Brackenbury, Shan Lu, and Blase Ur. "AutoTap: Synthesizing and Repairing Trigger-Action Programs Using LTL Properties". In: *Proceedings of the 41st International Conference on Software Engineering*. ICSE '19. Montreal, Quebec, Canada: IEEE Press, 2019, pp. 281–291. DOI: 10.1109/ICSE.2019.00043.

[155] Yuan Cao Zhang, Diarmuid Ó Séaghdha, Daniele Quercia, and Tamas Jambor. "Auralist: Introducing Serendipity into Music Recommendation". In: *Proceedings of the Fifth ACM International Conference on Web Search and Data Mining*. WSDM '12. Seattle, Washington, USA: ACM, 2012, pp. 13–22. ISBN: 978-1-4503-0747-5. DOI: 10.1145/2124295.2124300.

[156] Cai-Nicolas Ziegler, Sean M. McNee, Joseph A. Konstan, and Georg Lausen. "Improving Recommendation Lists Through Topic Diversification". In: *Proceedings of the 14th International Conference on World Wide Web*. WWW '05. Chiba, Japan: ACM, 2005, pp. 22–32. ISBN: 1-59593-046-9. DOI: 10.1145/1060745.1060754.

This Ph.D. thesis has been typeset by means of the TEX-system facilities. The typesetting engine was pdfLATEX. The document class was `toptesi`, by Claudio Beccari, with option `tipotesi=scudo`. This class is available in every up-to-date and complete TEX-system installation.