EUDoptimizer: Assisting End Users in Composing IF-THEN Rules Through Optimization

Original

Availability:
This version is available at: 11583/2728803 since: 2019-04-04T09:18:22Z

Publisher:
IEEE

Published
DOI:10.1109/ACCESS.2019.2905619

Terms of use:
openAccess
This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright
default
No description

(Article begins on next page)
EUDoptimizer: Assisting End Users in Composing IF-THEN Rules Through Optimization

FULVIO CORNO¹, (Member, IEEE), LUIGI DE RUSSIS¹, (Member, IEEE), and ALBERTO MONGE ROFFARELLO², (Student Member, IEEE).

¹Dipartimento di Automatica e Informatica, Politecnico di Torino, Torino, Italy (e-mail: fulvio.corno, luigi.derussis, alberto.monge@polito.it)

Corresponding author: Luigi De Russis (e-mail: luigi.derussis@polito.it).

ABSTRACT Nowadays, several interfaces for End-User Development (EUD) empower end users to jointly program the behavior of their smart devices and online services, typically through trigger-action rules. Despite their popularity, such interfaces often expose too much functionality, and force user to search among a large number of supported technologies disposed in confused grid menus. This paper contributes to the EUD with the aim of interactively assisting end users in composing IF-THEN rules with an optimizer in the loop. The goal, in particular, is to automatically redesign the layout of EUD interfaces to facilitate users in defining triggers and actions. For this purpose, a) we define a predictive model to characterize the composition of trigger-action rules on the basis of their final functionality; b) we adapt different optimization algorithms to explore the design space; and c) we present EUDoptimizer, the integration of our approach in IFTTT, one of the most popular EUD interfaces. We demonstrate that good layout solutions can be obtained in a reasonable amount of time. Furthermore, an empirical evaluation with 12 end users show evidence that EUDoptimizer reduces the efforts needed to compose trigger-action rules.

INDEX TERMS Combinatorial Optimization, End-User Development, Service Automation, Trigger-Action Programming

I. INTRODUCTION

The need of providing end users with adequate paradigms and tools to customize their smart environments on the basis of their personal needs has recently gained interest [1]. Nowadays, in fact, end users interact with a huge number of smart devices and online services on a daily base. End-User Development (EUD) methodologies [2] are suitable for letting non-technical users to program their smart objects and web services. In this field, the most commonly adopted EUD paradigm is trigger-action programming, with which users can define IF-THEN rules such as “if I receive a Facebook notification, then blink the Philips Hue lamp in the kitchen” or “if the Nest camera detects a movement, then set 22 degrees on my Nest thermostat.”

Contemporary EUD interfaces, e.g., IFTTT¹ and Zapier², are typically organized through grid layouts (Figure 1), i.e., a particular type of menu³ where items are organized in rows and columns. Furthermore, they share the same modality for composing trigger-action rules [1]: users have to firstly select a technology from a grid menu. The technology represents the manufacturer or the brand of a supported smart device or web service, e.g., Philips Hue or Facebook. On the selected technology, users can then define a particular trigger (if) through a wizard-based procedure. The same operations need to be repeated to define the action (then) of the rule.

Despite apparent simplicity, the composition of trigger-action rules through EUD interfaces is challenging, mainly because the rapid spread of new heterogeneous technologies [3]. Contemporary EUD interfaces display a huge number of information without any support to discover useful triggers and actions, and become too complex for non programmers. Consequently, users without technical skills may not find these systems useful [4]. IFTTT and Zapier, for example, force users to define their rules by searching

¹https://ifttt.com/, last visited on January 12, 2018
²https://zapier.com/, last visited on January 12, 2018
³In the remainder of the paper, grid layout and grid menu will be used interchangeably
between 500 and 1,000 supported technologies displayed with a meaningless order (Figure 1). Many previous works tried to mitigate such a complexity in different ways, thus confirming the need of better assisting users in programming their devices and services. Some of them, for example, face the problem by changing the underlying representations [5], [6], while others by exploring new composition paradigms, e.g., [1].

In this paper, we explore a different approach that is fully compatible with contemporary EUD interfaces, such as those of Figure 1, thus not requiring any radical change. Rather than acting on representations or composition paradigms, we adopt an optimizer in the loop to interactively assist end users in composing IF-THEN rules. The goal, in particular, is to dynamically redesign grid layouts in EUD interfaces in an interactive way, i.e., by considering the choices made by end users during the rule composition phase.

To reach our goal, we define two different predictive models to be used in a multi-objective optimization problem. In particular, we adapt 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 propose the Functionality Similarity Model (FSM), a novel model based on Semantic Web to take into account whether different and heterogeneous technologies provide similar functionality. As previous works demonstrate [7], [8], we claim that users would benefit from more support in discovering functionality, rather than being forced to get acquainted with technological details. Users, in fact, 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 EUD interfaces, we consider two different optimization algorithms, i.e., Simulated Annealing and Ant Colony Optimization, and we integrate those optimization methods in EUDoptimizer (End-User Development optimizer), an implementation of our approach on top of IFTTT. The implementation allowed us to qualitatively and quantitatively evaluate our approach. By exploiting a dataset of more than 200,000 trigger-action rules extracted from IFTTT [7], we show that EUDoptimizer can produce satisfactory solutions in a reasonable amount of time. Moreover, data from a user study with 12 participants suggest that EUDoptimizer can help end users define IF-THEN rules with less time and cognitive effort.

The paper is organized as follows. In Section II we contextualize our work by describing related works. In Section III we exemplify our approach by presenting a use case of EUDoptimizer. Then, we describe the components needed to realize such a use case, i.e., the predictive models (Section IV), the optimization methods (Section V), and the implementation of the EUDoptimizer tool that allows the interaction between users and the optimization method (Section VI). Section VII and Section VIII report a technical assessment of the implemented algorithm and the evaluation of EUDoptimizer in a user study, respectively. Eventually, Section IX discusses results and Section X concludes the paper.

II. BACKGROUND AND RELATED WORK

In this section, we first report some previous works that aim at improving EUD approaches and interfaces. Then, we contextualize our work in the large topic of applying optimization methods to user interface design.

A. IMPROVING END-USER DEVELOPMENT

Lieberman et al. [2] define End-User Development 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.” With the technological advances we are confronting today, people are increasingly moving from passive consumers to active producers of information, data, and software: in the last 10 years, several commercial tools for end-user personalization of devices and services, such as IFTTT or Zapier, were born.
Some previous works try to overcome the issues characterizing such tools, e.g., complexity [4] and scarce expressiveness [7], by acting on the underlying models, with the aim of simplifying the trigger-action rules composition process. Barricelli and Valtolina [5] present an extension of the trigger-action paradigm to better cope with the evolving Internet of Things (IoT) scenario. Besides devices and Web applications, they also incorporate other users, space and time, and the social dimension. With the EUPont ontology, Corno et al. [6] propose a high-level ontology for modeling triggers and actions. The semantic model allows the definition of generic rules that can be adapted for different contextual situations. In our work, we use EUPont in the predictive model used for assessing whether different technologies provide similar functionality.

Other works focus on new interfaces and tools for End-User Development. Desolda et al. [1], for example, report the results of a study to identify possible visual paradigms to compose trigger-action rules for smart environments, and present the architecture of a platform to support rules execution.

In our work, we follow a different approach. We are not interested in changing the underlying models nor the composition paradigms. To assist end users in defining IF-THEN rules, we employ optimization methods to dynamically redesign the grid menus of contemporary EUD interfaces. In a certain way, our approach can be seen as a sort of recommender system. In fact, by exploiting models of user performance and perception, we suggest technologies to be used in triggers and actions to the user, dynamically.

### III. OPTIMIZING IF-THEN RULES COMPOSITION

To clearly define and exemplify our approach, we introduce its implementation (EUDoptimizer, described in Section VI) in a 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.

**A. TRIGGER DEFINITION**

For the definition of the trigger, EUDoptimizer shows John a grid layout in which the optimizer has found a trade-off for a) displaying technologies according to their usage probability, and b) grouping together technologies to maintain logical groups of functional-related devices or services. Thanks to the FSM model, which characterize devices and services on the basis of what they can do, photo and video technologies with similar functionality are grouped together (as shown later in Figure 3a), thus facilitating John in finding what he needs. Furthermore, thanks to SDP and since most people extensively use such technologies, they are displayed on the top of the layout. John select the iOS Photo technology, that allows him to define a trigger to monitor every time he takes a photo on his iOS smartphone.
B. ACTION DEFINITION
The definition of the action strongly depends on the definition of the trigger, i.e., the grid menu for defining the action should interactively change according to the chosen trigger. As for the trigger definition, EUDoptimizer previously calculated an optimized components layout to be used as an initial solution by considering usage probabilities and functional similarities. As soon as John selects iOS Photo for defining the trigger, EUDoptimizer starts to refine the initial solution according to the user choice. The idea is to find the best trade-off for promoting the elements most commonly associated with the selected trigger technology, without affecting groups of functionally similar technologies. When John wants to define the action, EUDoptimizer shows the grid layout of Figure 4. Not surprisingly, the functionality in which John is interested are not uncommon. At the top of the layout, John can find different technologies 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 decide to define an action for saving the photos on his Google Drive 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.

IV. PREDICTIVE MODELS FOR TRIGGER-ACTION PROGRAMMING
The goal of our approach is to employ model-based optimization methods to dynamically redesign grid layouts in EUD 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 [17]. We adapt such a model to work with grid layouts in EUD interfaces, which display technologies for defining triggers and actions. Furthermore, similarly to the menu optimizer proposed by Bailly et al. [14], we use SDP in combination with a model that takes into account the expectation of item groups, i.e., the expectations of users in finding certain items together. Such a novel model for item grouping, named Functionality Similarity Model (FSM), considers similarities between technologies 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 Douglas blind for defining an action, because they both allow the increase or decrease of the brightnes 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 use the Semantic Web framework, and, in particular, the EUPont ontology.

A. 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’ laws, and integrates a transition from novice to expert performance. We adapt 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 selection time of an item $i$ in a menu with the following formula [17]:

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

where:
- $e_i$ models the user expertise with the item $i$.
- $T_{st}$ is the search time, i.e., the time to localize the item, linear with the total number of items when the user is inexperienced.
- $T_{dt}$ 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 $i$.
- $T_{pt}$ is the pointing time, i.e., the time to “point” the item, described according to the Fitts’ law, which predicts that items closer to the top are faster to select.

To predict the average performance of an entire menu, SDP uses the following formula:

$$T_{avg} = \sum_{i=1}^{n} p_i \cdot T_i, \quad (2)$$

where $p_i$ is the probability of item $i$ being selected, i.e., its usage probability, and $n$ is the number of menu items.

The probability function $p_i$ can be reformulated as the frequency probability, i.e., the probability of the technology $i$ to be used in a trigger or in an action. The idea is to move towards the top of the grid layout the technologies most commonly selected as triggers or actions. When the user selects a trigger $i$ for the trigger, the SDP model interactively changes in $SDP(i)$, where $p$ is reformulated as the bigram probability $p_{ij}$ of selecting an action technology $j$ after having selected the technology $i$ for the trigger. The idea is to move towards the top of the grid layout the action technologies most commonly associated with the trigger technology $i$.

The parameters we used for the SDP model are the same used both in the original paper [17] and in the optimizations carried out in MenuOptimizer [14].

B. FSM: FUNCTIONALITY SIMILARITY MODEL
We propose and define the Functionality Similarity Model to allow the optimizer to produce groups of technologies that are functionally correlated, even if they are heterogeneous technologies. The model exploits the EUPont [8] ontology, a semantic representation for trigger-action programming that offers a three-layer hierarchical categorization of triggers and actions in terms of their final functionality. The three categories, named Low-Level, Medium-Level, and High-Level, represent three different levels of abstraction in which triggers and actions can be expressed and categorized. For example, the Low-Level category “turn lamp x on” includes

---

4https://elite.polito.it/ontologies/eupont.owl, last visited on April 10, 2018
all the actions aiming at turning a specific lamp (identified by \( x \)) on. Such a category can be further abstracted in “turn the lights on” (Medium-Level category) and “illuminate a place” (High-Level category), respectively. With this hierarchical representation, the “illuminate a place” category includes all the actions for turning lights on, along with other actions that allow to reach the final goal of illuminating a place, e.g., opening a blind.

By considering this EUPont categorization, the Trigger Functionality Association (\( FA_t \)) between two technologies \( i \) and \( j \) is calculated as:

\[
FA_t(i, j) = \alpha_f \cdot LL_t(i, j) + \beta_f \cdot ML_t(i, j) + \gamma_f \cdot HL_t(i, j),
\]

where:

- \( LL_t(i, j) \) is the number of Low-Level categories for which \( i \) and \( j \) both offer at least one trigger.
- \( ML_t(i, j) \) is the number of Medium-Level categories for which \( i \) and \( j \) both offer at least one trigger.
- \( HL_t(i, j) \) is the number of High-Level categories for which \( i \) and \( j \) both offer at least one trigger.
- \( \alpha_f, \beta_f, \) and \( \gamma_f \) 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:

\[
FA_a(i, j) = \alpha_f \cdot LL_a(i, j) + \beta_f \cdot ML_a(i, j) + \gamma_f \cdot HL_a(i, j)
\]

In our implementation, we empirically set \( \alpha_f = 0.6 \), \( \beta_f = 0.3 \), and \( \gamma_f = 0.1 \). This means that technologies that shares Low-Level (very specific) categories are more similar than technologies that only shares Medium or High-Level categories, that are intrinsically more abstract.

To use the model in a minimization problem, we exploit 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 \( FI_a \)):

\[
FI_t = \sum_{i=1}^{n} \sum_{j=i+1}^{n} FA_t(i, j) \cdot d(i, j),
\]

\[
FI_a = \sum_{i=1}^{n} \sum_{j=i+1}^{n} FA_a(i, j) \cdot d(i, j),
\]

where \( d(i, j) \) is the euclidean distance between objects \( i \) and \( j \) in the grid layout.

As shown in Figure 3b, the FSM model has desirable effects on the optimizer: technologies that offer triggers (or actions) with similar functionality tend to be pulled together, while unrelated technologies are moved away.

**V. OPTIMIZATION PROBLEM AND METHODS**

In this section, we first formulate the optimization problem by defining the objective function we used to explore the design space. Then, we present the optimization methods we used to attack the problem, based on metaheuristic strategies.

**A. PROBLEM FORMULATION**

To explore the design space looking for “good” or “desirable” grid menu alternatives, we define a multi-objective task. The goal of the optimizer is to minimize a weighted combination of the outputs of the the two models \( M_t \) exploited by EU-Doptimizer, i.e., SDP and FSM:

\[
\min \sum_{i=1}^{2} \lambda_i \cdot M_i,
\]

where the sum of all the weights \( \lambda_i \) is 1. In our implementation, we empirically set the \( \lambda \) values thanks to the trials performed in the technical assessment of the implemented algorithms (Section VII).

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

\[
\theta_i = M_i(x_0).
\]

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

\[
\min (\lambda_1 \cdot SDP + \lambda_2 \cdot FI_t)
\]

while for the action the problem is:

\[
\min (\lambda_1 \cdot SDP + \lambda_2 \cdot FI_a)
\]

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

\[
\min (\lambda_1 \cdot SDP_i + \lambda_2 \cdot FI_a)
\]

**B. SOLVING THE OPTIMIZATION PROBLEM**

The problem of designing menu systems, both linear and grid-based, can be formulated as a Quadratic Assignment Problem (QAP) [14]. Developed in operational research, QAP [18] 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 [19].

Our implementation, described in the next section, supports two metaheuristics successfully used for QAP problems, i.e., Simulated Annealing [20], [21] and Ant Colony System [22]. 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.

Algorithm 1 Simulated Annealing
1: Let $s = s_0$
2: for $k = 0$ through $k_{\text{max}}$ do
3: \hspace{1em} $T \leftarrow \text{temperature}(k/k_{\text{max}})$
4: \hspace{1em} Pick a random neighbour, $s_{\text{new}} \leftarrow \text{neighbour}(s)$
5: \hspace{1em} if $P(E(s), E(s_{\text{new}}), T) \geq \text{random}(0, 1)$ then
6: \hspace{2em} $s \leftarrow s_{\text{new}}$
7: Output: the final state $s$

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.

Algorithm 2 Ant Colony System
1: while (not converged) do
2: \hspace{1em} Position each ant in a starting node
3: \hspace{2em} repeat
4: \hspace{3em} for all ant do
5: \hspace{4em} Chose next node with the transition rule
6: \hspace{4em} Apply local pheromone update
7: \hspace{2em} until every ant has built a solution
8: Update best solution
9: Apply global pheromone update

VI. EUDOPTIMIZER
In this section, we describe EUDoptimizer, an implementation of our approach on top of IFTTT. Despite 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 [7]. To maintain a high level of interactivity, we implemented a client-server architecture between the optimizer and the user interface for composing trigger-action rules.

A. DATA AND MODELS
We exploited a large dataset of trigger-action rules obtained by Ur et al. [7] with a web scrape of the public rules shared on IFTTT as of September 2016. The dataset is composed of 295,156 rules created by 129,206 different authors, and it includes more than 200 different technologies, ranging from smart home devices to web applications. Besides the information about triggers and actions, each rule also includes a sharing information, i.e., the number of times that it has been reused by other users, thus providing a direct measure of its popularity. We used the sharing information of each rule, normalized between 0 and 1, to calculate the frequency probabilities and the bigram probabilities, to be used in SDP model. Furthermore, we linked each technology with the corresponding EUPoint entity by using the translation process described in [6], and we calculated the Functionality Associations (FA) to be used in the FSM model.

B. USER INTERFACE
By exploiting the information extracted from the dataset, we modeled a user interface after IFTTT with the AngularJS framework\(^5\). The interface, shown in Figure 2, allows the composition of trigger-action rules exactly as in IFTTT. For defining a trigger, for example, users have to click on the “this” button (Figure 2a), and then select the desired technology from a grid layout (Figure 2b). Finally, they can select the specific trigger to be monitored (Figure 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 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.

C. OPTIMIZER
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, we empirically run a set of 100 optimizations by varying the parameter values. Both optimizers provide the same functionality. They initially generates two grid layouts T1 and A1 (for defining triggers and actions, respectively) by using the two “static” versions of the optimization problem (Equation 9 and Equation 10). 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 technology for defining the trigger, each optimizer interactively receives the information. Starting from the layout A1, each of them starts to explore the problem described by the Equation 11 to refine the initial layout to take into account the user’s choice. When the user finishes the trigger definition, i.e., she completes it with all the required parameters, each optimizer generates a grid menu A2, an improved version of A1 in which the technologies that are most likely to be used with the selected trigger are promoted towards the top.

VII. TECHNICAL ASSESSMENT
To assess the feasibility of our approach, and to evaluate which optimizer provide better solutions, we executed the SA and the ACS solvers “off-line,” by changing the number of

\(^5\)https://angularjs.org/, last visited on February 12, 2018
iterations of the algorithms. During the trials, we also tuned the parameters of the algorithms, along with the weights of our optimization problem.

A. TRIGGER DEFINITION

We first executed the optimizers for calculating a grid layout for trigger definition, thus solving Equation 9. Table 1 and Table 2 report the results obtained with 100, 1,000, 5,000, and 10,000 iterations with SA and ACS, respectively. Despite the ACS solver provides better solutions for 100 and 1,000 iterations, the SA solver performs better with a higher number of iterations. Furthermore, SA is faster than ACS in all cases. Figure 3 shows two screenshots of the grid menu calculated with 10,000 iterations of SA, obtained in less than 20 minutes.

**TABLE 1:** Results of the Simulated Annealing for the trigger layout with 100, 1,000, 5,000, and 10,000 iterations.

<table>
<thead>
<tr>
<th>Iterations</th>
<th>Time [sec]</th>
<th>Obj. value</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>12</td>
<td>0.99</td>
</tr>
<tr>
<td>1,000</td>
<td>106</td>
<td>0.98</td>
</tr>
<tr>
<td>5,000</td>
<td>536</td>
<td>0.47</td>
</tr>
<tr>
<td>10,000</td>
<td>1,031</td>
<td>0.43</td>
</tr>
</tbody>
</table>

**TABLE 2:** Results of the Ant Colony System for the trigger layout with 100, 1,000, 5,000 and 10,000 iterations.

<table>
<thead>
<tr>
<th>Iterations</th>
<th>Time [sec]</th>
<th>Obj. value</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>178</td>
<td>0.89</td>
</tr>
<tr>
<td>1,000</td>
<td>1,896</td>
<td>0.88</td>
</tr>
<tr>
<td>5,000</td>
<td>10,012</td>
<td>0.86</td>
</tr>
<tr>
<td>10,000</td>
<td>20,515</td>
<td>0.82</td>
</tr>
</tbody>
</table>

We can observe that:

- The 10 most frequently used technologies\(^6\) for defining triggers are prominently placed in the 10 positions closer to the top of the grid menu (Figure 3a). Their *functional* similarities are pulled together and organized in logical groups, e.g., locations (*Android Location*, *iOS Location*), photos and videos (*iOSPhoto, Android Photo, Eyefy, Youtube, Flickr, Dailymotion*, and *500px*), etc.
- Figure 3b further highlights that technologies with functional similarities are pulled together in logical groups. The figure, in particular, shows a huge group of hubs, cameras, and doorbells, all related to home security.

B. ACTION DEFINITION

We then executed the optimizers for calculating the action grid menu. In this case, we initially used the solvers to calculate (with 10,000 iterations) an initial solution for the menu, i.e., by solving Equation 10. Then, we manually fixed the selected trigger technology to *iOS Photo*, one of the most used trigger in the data set, and we tested the optimizers to solve the problem defined by Equation 11, i.e., the interactive optimization. Table 3 and Table 4 report the optimization results for SA and ACS, respectively. Also in this case, SA performed better than ACS with 10,000 iterations. Figure 4 shows the top part of the best grid menu obtained with 10,000 iterations by SA. Both Figure and Table confirm that the SA solver produces good solutions in a reasonable amount of time. In fact, 9 of the 10 technologies most frequently associated with *iOS Photo* are presented on the top of the grid layout. Furthermore, there are logical groups of related technologies that allow the definition of similar functionality, e.g., *One Note, Nimbus Note*, and *Evernote*.

**TABLE 3:** Results of the Simulated Annealing for the action layout when the selected trigger technology is *iOS Photo*.

<table>
<thead>
<tr>
<th>Iterations</th>
<th>Time [sec]</th>
<th>Obj. value</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>10</td>
<td>1.01</td>
</tr>
<tr>
<td>1,000</td>
<td>91</td>
<td>0.98</td>
</tr>
<tr>
<td>5,000</td>
<td>468</td>
<td>0.47</td>
</tr>
<tr>
<td>10,000</td>
<td>895</td>
<td>0.45</td>
</tr>
</tbody>
</table>

VIII. USER STUDY

Thanks to the technical assessment, we selected the SA 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 compose trigger-action rules
Corno et al.: EUDoptimizer: Assisting End Users in Composing IF-THEN Rules Through Optimization

FIGURE 3: Optimized grid layout for defining triggers calculated with the Simulated Annealing solver (10,000 iterations). (a) shows the top of the layout (the added stars indicate the 10 most frequently used technologies). (b) shows the bottom part; a yellow border highlights a logical group of technologies related to home security.

TABLE 4: Results of the Ant Colony System for the action layout when the selected trigger technology is iOS Photo.

<table>
<thead>
<tr>
<th>Iterations</th>
<th>Time [sec]</th>
<th>Obj. value</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>171</td>
<td>0.88</td>
</tr>
<tr>
<td>1000</td>
<td>1669</td>
<td>0.83</td>
</tr>
<tr>
<td>5000</td>
<td>11547</td>
<td>0.82</td>
</tr>
<tr>
<td>10000</td>
<td>20515</td>
<td>0.82</td>
</tr>
</tbody>
</table>

FIGURE 4: Optimized grid layout for actions calculated with SA when the selected trigger technology is iOS Photo (10,000 iterations). Stars indicate the most frequently used technologies: 9 out of 10 are at the top of the grid layout.

with both interfaces. The goal was to understand whether EUDoptimizer a) improved the user performance, i.e., the time needed for composing trigger-action rules, and b) reduced the cognitive load in the composition of trigger-action rules with respect to the “normal” version. In this section, we describe the study, and we report the quantitative and qualitative results used for investigating the time variable and the cognitive effort, respectively.

A. STUDY DESCRIPTION

1) Design and Procedure
We devised a within-subject user study, where we considered the interface version (IFTTT vs. EUDoptimizer) as the independent variable. We first submitted 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 composing a trigger-action rule in IFTTT as an example. After the training phase, we asked participants to complete 6 similar tasks related to the composition 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.

2) 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 ($M = 1.67, SD = 0.89$).

3) Tasks
In the study, each task consisted in the composition of a single trigger-action rule. We defined 6 different tasks that asked participants to replicate trigger-action rules previously...
extracted from the IFTTT dataset [7]. To explore the full range of possible alternatives, e.g., to evaluate EUDoptimizer both with commonly and rarely used technologies, 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.

4) **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 technology to define the trigger from the grid layout.
- **Action Time (AT).** The time for selecting the technology to define the action from the grid layout.
- **Rule Time (RT).** The time for composing the entire rule, composed of the Trigger and Action times (TT and AT), and the time needed for completing the specific trigger and action, e.g., for filling the required details.

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

**B. QUANTITATIVE RESULTS**

Table 5, Table 6, and Figure 5 show and compare the measures obtained with the IFTTT and the EUDoptimizer interfaces, respectively, averaged for all participants and tasks.

**TABLE 5: Average Trigger Time (TT), Action Time (AT), and Rule Time (RT) with the IFTTT interface.**

<table>
<thead>
<tr>
<th>Measure</th>
<th>Mean [sec]</th>
<th>Std. Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>TT</td>
<td>37.29</td>
<td>13.44</td>
</tr>
<tr>
<td>AT</td>
<td>20.87</td>
<td>11.18</td>
</tr>
<tr>
<td>RT</td>
<td>81.77</td>
<td>23.14</td>
</tr>
</tbody>
</table>

All the time measures were lower with the optimized interface. Therefore, EUDoptimizer improved the selection time in the trigger and in the action definition. Such improvements were reflected in the time needed by end users to compose a trigger-action rule (RT): the EUDoptimizer interface, in fact, allowed participants to define rules faster than the IFTTT interface.

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 analyses. 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 composition time of trigger-action rules.

We can say that EUDoptimizer improves the composition of trigger-action rules by reducing the time effort in the definition of triggers and actions. To further demonstrate such a statement, we separately analyzed the measures for the two uncommon rules, only (Figure 6). We were particularly interested in evaluating the potential of EUDoptimizer in the worst case. In fact, since we used the dataset to weight items by their frequency, technologies involved in the uncommon rules were not placed in the first position of the grid layouts.

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 ($75.32 \pm 44.25$ sec vs. $116.81 \pm 49.19$ sec). On average, the selection of a technology for defining actions was in-
steady performed with similar performances by participants (16.67 ± 24.21 sec vs. 17.70 ± 33.43 sec). However, the time for composing the entire rules (RT measure) was considerably lower with the EUDoptimizer interface (116.52 ± 59.82 sec vs. 158.31 ± 53.17 sec).

C. QUALITATIVE RESULTS
Qualitative data extracted from the audio recorded files show that the benefit 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 technologies by ordering the components layout with logical groups of functional-related elements. 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 technologies 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 technologies. 5 participants were happy of the “logical” groups of technologies 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 technologies 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.”

X. CONCLUSIONS
In this paper we investigated a new approach to interactively assist end users in composing IF-THEN rules with an optimizer in the loop. The goal was to dynamically redesign layouts in EUD interfaces in an interactive way, i.e., by considering the choices made by users. To reach our goal, we adapted a state-of-the-art predictive model of user performance in menu search, and we define a novel model to organize technologies (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 in EUDoptimizer, an implementation of our approach on top of IFTTT. The implementation showed that the optimization can produce satisfactory solutions in a reasonable amount of time. Moreover, a user study with 12 participants confirmed that EUDoptimizer reduces the time and cognitive efforts needed by end users to compose trigger-action rules. There are several opportunities to improve the approach. The models, for example, could be further analyzed: are there any other criteria, beside the functionality similarity, to refine the optimization approach? Which criteria has a greater impact on end user efforts? Such considerations are guiding our current works.
REFERENCES


