POLITECNICO DI TORINO
Repository ISTITUZIONALE

A High-Level Semantic Approach to End-User Development in the Internet of Things

Original
A High-Level Semantic Approach to End-User Development in the Internet of Things / Corno, F., De Russis, L., Monge Roffarello, A.. - In: INTERNATIONAL JOURNAL OF HUMAN-COMPUTER STUDIES. - ISSN 1071-5819. - STAMPA. - 125:(2019), pp. 41-54. [10.1016/j.ijhcs.2018.12.008]

Availability:
This version is available at: 11583/2720712 since: 2019-01-07T12:26:48Z

Publisher:
Elsevier

Published
DOI:10.1016/j.ijhcs.2018.12.008

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright
Elsevier postprint/Author's Accepted Manuscript

© 2019. This manuscript version is made available under the CC-BY-NC-ND 4.0 license
http://creativecommons.org/licenses/by-nc-nd/4.0/.The final authenticated version is available online at:
http://dx.doi.org/10.1016/j.ijhcs.2018.12.008

(Article begins on next page)

18 November 2026

A High-Level Semantic Approach to End-User Development in the Internet of Things

Fulvio Corno[a], Luigi De Russis[a,*], Alberto Monge Roffarello[a]

[a] *Department of Control and Computer Engineering, Politecnico di Torino, 10129 Torino, Italy*

Abstract

Various programming environments for End-User Development (EUD) allow the composition of Internet of Things (IoT) applications, i.e., connections between IoT objects to personalize their joint behavior. These environments, however, only support a one-to-one mapping between pairs of object instances, and adopt a low level of abstraction that forces users to be aware of every single technology they may encounter in their applications. As a consequence, numerous open questions remain: would a "higher level" of abstraction help users creating their IoT applications more effectively and efficiently compared with the contemporary low-level representation? Which representation would users prefer? How high-level IoT applications could be actually executed? To answer these questions, we introduce EUPont, a high-level semantic model for EUD in the IoT. EUPont allows the creation of high-level IoT applications, able to adapt to different contextual situations. By integrating the ontology in the architecture of an EUD platform, we demonstrate how the semantic capabilities of the model allow the execution of high-level IoT applications. Furthermore, we evaluate the approach in a user study with 30 participants, by comparing a web interface for composing IoT applications powered by EUPont with the one employed by a widely used EUD platform. Results show that the high-level approach is understandable, and it allows users to create IoT applications more correctly and

*Corresponding author

Email addresses: fulvio.corno@polito.it (Fulvio Corno), luigi.derussis@polito.it (Luigi De Russis), alberto.monge@polito.it (Alberto Monge Roffarello)

quickly than contemporary solutions.

Keywords: End-User Development, Internet of Things, Trigger-Action Programming, Semantic Web, Abstraction

1. Introduction and Problem Definition

The Internet of Things (IoT) is, nowadays, a well-established paradigm [1]. In this context, end-user personalization of IoT systems is particularly important, as demonstrated by many works in the literature [2, 3, 4]. End-User Development (EUD), in particular, empowers users to define IoT applications, i.e., connections between different IoT devices and Web services, in various areas, like the home, the car, or for a healthy lifestyle. Many works in the literature (e.g., [5, 2, 6, 3, 4]) demonstrate the effective applicability of EUD techniques for the creation IoT applications. Such personalizations can be defined, nowadays, through different cloud services. These services typically employ a trigger-action programming approach (e.g., *"IF something happens, THEN perform an action"*) to combine different IoT devices and Web services. With this approach, users can connect a pair of "things" in such a way that, when one of them performs a particular action, another action is automatically triggered on the second. IoT applications created with this approach are also called *trigger-action rules*, and present a one-to-one mapping between pairs of service or device instances. They may include, for example, "if the bedroom motion sensor detects a movement, then turn on the table lamps in the bedroom" or "if tomorrow's forecast calls for rain, then delay the garden sprinkler system." End users that want to realize such applications do not need to write any code; instead, they can take advantage of Task Automation (TA) tools [7] such as IFTTT[1] or Zapier[2]. Among these tools, IFTTT is the most popular, and has drawn a large community of users. IFTTT allows the composition of IoT applications (named *recipes*) between more than 500 supported devices and services

[1] `https://ifttt.com` (last visited on January 11, 2018)

[2] `https://zapier.com/` (last visited on January 11, 2018)

(named *channels*)[3].

TA environments like IFTTT are successful in terms of user understandability and ease of use, but present their own set of problems. We identify, in particular, two major issues:

- *Adaptation.* In the forthcoming IoT world, new "things" will not always be knowable a priori [8] but they may appear and disappear at every moment, also depending on user location (e.g., as with public services in a smart city). Contemporary TA tools work only with well-know IoT devices, previously associated to a specific user, only.

- *Abstraction.* Two IoT devices that provide equivalent or identical functions (e.g., setting the indoor temperature) but differ in brands are treated like distinct entities. As the number of available IoT devices grows and varies, the complexity of the IoT ecosystem becomes higher and higher [9]. Therefore, the amount of information present in current TA tools can become too high and the interfaces cluttered.

With such a "low-level" of abstraction, the user experience with contemporary TA tools is put to a hard test. First, users are forced to know in advance any involved technological detail, and they have to define several rules to program their IoT ecosystem, i.e., every IoT devices and Web service needs to be managed separately. For instance, a user cannot create an IoT application that can be applied to all her connected lamps, unless they are equally branded, nor to other kinds of devices that may provide interior lighting. Furthermore, contemporary IoT application cannot include friends' or family's availability or their GPS positioning, and users cannot define similar rules for yet unknown places or "thing".

To overcome these issues, we claim that a new breed of programming platforms should be designed to support a "higher level" representation of trigger-

[3]On November 2016, IFTTT rebranded its recipes as applets and its channels as services. We maintained the older names to better discuss related work.

action rules (and involved devices and services). However, different open questions still remain: would a higher level of abstraction help users creating their IoT applications more effectively and efficiently compared with the contemporary low-level representation? Which representation would users prefer, and which advantages and disadvantages would they perceive? How high-level IoT applications could be actually executed? To answer these questions, we first formally define EUPont (**E**nd-**U**ser **P**rogramming **ont**ology) [10], a high-level representation for End-User Development that allows users to model abstract IoT applications like *"if I enter a closed space, then set the temperature to 20 Celsius degree."* Such applications can be adapted to different contextual situations, independently of manufacturers, brands, and other technical details. The representation abstracts the IoT ecosystem by modeling devices and services on the basis of their functionality. Through semantic and reasoning capabilities, 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. After defining the model, we describe its integration in the architecture of an EUD platform, and we explore the advantages of using such a representation in the definition of IoT applications thanks to a user study with 30 participants. Results confirm the suitability and understandability of the approach, and provide interesting insights. They also show that the EUPont approach allows users to create IoT applications more correctly and quickly than contemporary solutions.

The remainder of the paper is organized as follows: Section 2 reviews the related literature. Section 3 describes the design and organization of the EUPont ontology, while Section 4 presents the architecture of the platform that exploits the ontology for the definition and the execution of trigger-action rules. Section 5 describes the user study we conducted to evaluate the proposed approach and the representation. Section 6 discusses the results of the user study. Eventually, Section 7 concludes the paper.

2. Related Works

2.1. End-User Development in the Internet of Things

According to Lieberman et al. [11], End-User Development 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*". 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 [12]. They can access new building blocks and tools, analogously to what happened with blogs and wikis during the early phases of the Web [13]. Moreover, integration of IoT technologies with Web services and applications through end-user programming environments allows users to effectively participate in the IoT [14]. Therefore, is not surprising that, in the last years, and several commercial Task Automation (TA) tools (e.g., IFTTT or Zapier) were born with the aim of allowing end user personalization of IoT devices and Web services.

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 [5], a visual, PC-based, and rule-based system for building context-aware applications that does not require users to write any code. To compose a context-aware application with iCAP, users employ the trigger-action approach: they drag components (i.e., devices, locations, time, etc.) either in a "situation" section (*if*) or in an "action" area (*then*). Components can be already present in the system or can be created by users by sketching their icon. More recent works move away from the desktop to embrace mobile or Web-based interfaces, but without substantially changing the underlying programming paradigm.

The trigger-action programming approach is, in fact, one of the most popular adopted paradigms and offers a very simple and easy to learn solution for creating end-user IoT applications, according to Barricelli and Valtolina [9]. They

analyzed the most popular end-users tools for the creation of IoT applications, and presented a classification model for the IoT. They found two main types of task automation environments, which differ in terms of activities and interaction styles. The first type included programming environments like IFTTT, i.e., tools that allow users to define sets of desired behaviors in response to a specific event (e.g., [3, 4]). The second type stemmed from the (now discontinued) Yahoo Pipes, i.e., tools that use formula languages and/or visual programming for data transformation and mashup (e.g., [15]). They concluded the analysis by arguing that the second type of programming environments "offers a too complex solution for supporting end-users in expressing their preferences." To better cope with the evolving IoT scenario, Barricelli and Valtolina did not move towards a high level representation, instead they presented an extension of the trigger-action paradigm that incorporated not only devices, sensors, and Web applications, but also recommendation systems, other IoT users, space and time, and the social dimension.

Similarly, Ur et al. [2] found that the trigger-action approach can be both useful and usable for end-user development in IoT settings like smart homes. 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 discovered that inexperienced users can quickly learn to create programs with multiple triggers and actions. They also found that the level of abstraction end-users employ to express triggers needs to be better explored. In particular, they found that many users expressed 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. [16] empirically analyzed more than 200,000 IFTTT public recipes, the largest-scale investigation of this type up to now, finding that a large number of users is crafting a diverse set of IoT applications, which represents a very broad array of connections for filling gaps in devices and services functionality. According to the authors, this explosion of channels and

connections highlights the need to provide users with more support for discovering functionality and managing collections of IoT applications. The analysis emphasizes also the future need of making "IFTTT recipes more expressive."

Huang and Cakmak [17] 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. After a characterization of these issues, they 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."

Our work starts from the issues and the opportunities presented by these papers and explores the need and the advantages of describing IoT applications with a higher level of abstraction. For this purpose, we adopted a semantic approach by designing the EUPont ontological model.

2.2. Modeling the IoT Ecosystem Through Ontologies

Adding semantics to IoT systems is a topic of particular interest in the literature. The lack of open IoT standards naturally leads to a semantic-oriented perspective [18], and researchers agree that the IoT could benefit from a semantic approach in terms of interoperability, data integration, and knowledge extraction [19]. 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) [20], 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 [21]. 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. [22], 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 [22], 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 include in their models other concepts, such as users, on-line services, interfaces, etc. In the IOT-A project[4], the authors identified entities, resources, and services as key concepts of the IoT domain. In [23], the authors proposed a modeling approach in which IoT resources are able to expose standard service interfaces. Similarly, Wang et al. [24] 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 IoT 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 homes. With DogOnt [25], the authors presented a building 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[5] organization developed

[4] *http://www.iot-a.eu/public* (last visited on November 11, 2016)
[5] *https://www.tno.nl/en/* (last visited on November 11, 2016)

SAREF[6], a shared model of consensus that facilitates the matching of existing assets (standards, protocols, data-models, etc.) in the smart appliances domain.

2.3. Tools and Models for Context-Dependent Applications

The process of abstracting for lowering the complexity is already adopted in many different fields such as software engineering [26], and the usage of semantic methodologies is a common approach for reaching this purpose. In the field of IoT personalization, for example, Ardito et al. [27] presented a visual composition paradigm that allows non-programmers to synchronize the behavior of smart objects by defining their semantics, with the aim of determining more engaging user experiences in Cultural Heritage (CH) sites. The composition metaphor, in particular, promotes smart objects as entities characterized not only by native events and actions (as conceived in many IoT platforms) but also by attributes that the domain experts can define to assign semantics to the objects. Differently from such an approach, that requires users to know in advance what a device can do, we formalize the semantics of the IoT devices and services *in advance*, i.e., by designing the EUPont ontology.

To support the high-level abstraction modeled by the EUPont ontology, EUD platforms must be able to adapt the same IoT application to different contextual situations. Starting from the Context Toolkit [28], there has been a long history of interest in supporting the development of context-dependent applications. Gu et al. [29], for example, developed a Service Oriented Context-Aware Middleware, named SOCAM, for building context-dependent services. As in our work, SOCAM used ontologies and semantic reasoning to provide efficient support for acquiring, discovering, interpreting and accessing various contexts. In particular, the middleware model relied on two-levels context ontologies, where the upper ontology defined general concepts, while the bottom ontologies dealt with low-level domain-oriented contexts (e.g., smart home, vehicle, etc.). This decoupling of layers is particularly important to tackle domain specificity, as

[6] *http://ontology.tno.nl/saref* (last visited on November 11, 2016)

also reported by Desolda et al. [30]. In their work, the authors proposed a model that includes new operators for defining rules, by combining multiple events and conditions exposed by smart objects. Besides reporting the results of a study to identify possible visual paradigms to compose their rules, the authors 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. We also adopted a layered approach in our work, both in the architecture of the EUD platform and in the EUPont ontology. In our model, high-level concepts (i.e., the abstract triggers and actions) are separated from low-level concepts (e.g., real devices and services), and they are linked together thanks to the semantic and reasoning capabilities of the model.

Another system for managing the context was proposed in the work of Indulska and Robinson [31]. Here, the authors proposed a model-based context management system, named ACoMS, that could dynamically configure and reconfigure the gathering and preprocessing functionality of contextual information, to provide fault tolerant provisioning services. In the same work, the authors classified the development of context-dependent applications according to three main approaches:

- *no application-level context model*: each application directly communicates with sensors and other sources of context information;
- *implicit context model*: applications are developed with the aid of reusable libraries/toolkits for processing context information;
- *explicit context model*: applications have their own well-defined context models and use a shared context management.

In our work, we follow a mixed approach between *implicit* and *explicit context model*. In fact, the proposed EUD platform shares a generic context model, that is the result of a merging process between different specific context infor-

mation. Such specific information are relative to a particular application area, and can be retrieved thanks to reusable libraries and toolkits. For example, in Section 4, we use the Dog Gateway [32, 33]. The Dog Gateway is a home and building automation gateway able to manage different networks as a single, technology neutral, home automation system. Similarly to our work, but following the *explicit context model* approach, Ghiani et al. [4] proposed a method and a set of tools that allow 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. Differently from that work, where trigger-action rules maintain a direct mapping with real devices and services, we designed EUPont with the possibility of expressing more abstract IoT applications.

3. The EUPont Ontology

To design the EUPont ontology, we carefully reviewed the content and the structure of existing TA tools in the IoT as well as the reported issues and challenges from the literature. In particular, we deeply analyzed all the IFTTT channels, along with their 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 the temperature*" by grouping different actions of 20 diverse devices (e.g., Caleo, ecobee, Nest, tado Smart AC Control, Wink Aros, etc.). Starting from the result of this analysis and taking into account the findings from related works, we formally designed EUPont to validate our hypothesis. EUPont allows end-users to define abstract and technology-independent trigger-action rules that can be adapted to different contextual situation. Furthermore, thanks to semantic and reasoning capabilities, the ontology itself provides a strong support for executing the defined trigger-action rules on real devices and services. We chose to exploit the Semantic Web frame-

work 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 "smart objects" can be easily integrated in existing end-user development solutions. Semantic technologies offer *by nature* advantages in terms of data reuse and integration, thus making it easy to integrate new devices and 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 IoT devices or services can perform a particular action?"* or *"which IoT devices or services can generate a particular event?"*, thus easily implementing a "programming by functionality" feature.

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.

EUPont models all the 379 devices and services (100%) available on IFTTT on March 2017. We also manually mapped in the EUPont representation 951 IFTTT triggers out of a total of 976 (97.44%), and 528 actions out of the 551 (95.83%) available on IFTTT. Triggers and actions that we could not model in the EUPont representation were ambiguous in terms of functionality. For example, the action *"execute a scene with my IntesisHome hub"* is ambiguous because the scene that can be activated on the hub is defined by the user, and it can include many different settings. Moreover, to investigate the completeness of the obtained model, we automatically translated the 295,156 IFTTT

rules present in the repository extracted by Ur et al. [16] into the EUPont representation. We were able to represent 290,964 IFTTT rules, thus reaching a percentage coverage of 98.58%, as better detailed in [10].

To develop the EUPont ontology, we used Protégé[7], a free and open-source OWL ontology editor for building intelligent systems. Moreover, we adopted HermiT[8] as the semantic reasoner. The resulting ontology is available at `http://elite.polito.it/ontologies/eupont.owl`.

3.1. EUPont structure

Figure 1 shows the general structure of EUPont, while Table 1 reports the main modeled concepts and how they are linked together. The ontology is composed of three layers, which are interlinked to support both the composition and the execution of abstract and technology-independent trigger-action rules. In particular, the Trigger-Action Programming layer (TAP) allows end-users to compose rules in the EUPont representation, while the IoT Ecosystem (IoT) hides the technological details of real devices and services. In addition, the shared Contextual Information layer (CI), along with a set of SWRL rules, supports the semantic reasoning to provide a mapping between real devices and services with the aim of abstracting the triggers and actions they can reproduce.

The **Contextual Information (CI) Layer** describes the concepts of Location(s) and User(s) that are shared between the TAP and the IoT layers. Such concepts allow us to represent Locations and Users that can be referred by trigger-action rules, and describe the contextual information of the IoT ecosystem, e.g., the position of a device, or the users subscribed to a Web service.

The **Trigger-Action Programming (TAP) Layer** allows the definition of abstract trigger-action rules, that are independent from manufacturers, brands, or any other technological details. As reported in Table 1, the layer models three main concepts, i.e., Rule, Trigger, and Action. A Rule can have multiple

[7] `https://protege.stanford.edu` (last visited on January 11, 2018)
[8] `http://www.hermit-reasoner.com/` (last visited on January 11, 2018)

Table 1: Main concepts modeled in the EUPont ontology.

Concept	Description	Layer
Rule	The EUPont representation for describing a rule. A Rule has at least one Trigger and at least one Action.	TAP
Trigger	The EUPont representation for a Trigger, i.e., an event to react to. It is specialized in a hierarchy of events expressed at different level of abstraction, and can be linked with Location(s) and User(s), i.e., the Contextual Information.	TAP
Action	The EUPont representation for an Action, i.e., an action to perform. It is specialized in a hierarchy of actions expressed at different level of abstraction, and can be linked with Location(s) and User(s), i.e., the Contextual Information.	TAP
Location	The EUPont representation for Location(s) in the modeled ecosystem.	CI
User	The EUPont representation for User(s) in the modeled ecosystem.	CI
IoT Entity	The EUPont representation for IoT devices or services, that are modeled on the basis of their categories (e.g., environment systems, user devices, etc) and final capabilities, i.e., offered Service(s). IoT Entities can be linked with Location(s), e.g., for specifying the position of a device, and User(s), e.g., for specifying who is authorized to control them.	IoT
Service	The EUPont representation for one of the capabilities of an IoT Entity.	IoT
Command	The EUPont representation for a low-level action that can be executed on an IoT Entity that offers a particular Service. It includes the technology-specific features, i.e., the low-level details, needed to interface the IoT entitiy.	IoT
Notification	The EUPont representation for a low-level event that can be generated by an IoT Entity that offers a particular Service. It includes the technology-specific features, i.e., the low-level details, needed to interface the IoT entitiy.	IoT

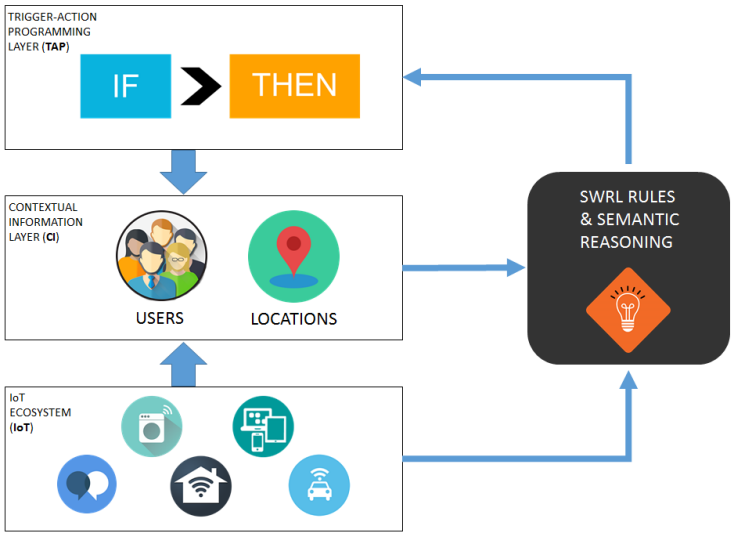


Figure 1: The EUPont structure: the Trigger-Action Programming and the IoT Ecosystem layers refer to the same Contextual Information, and are linked together thanks to a set of SWRL rules.

Triggers and multiple Actions [30].

To allow end-users to choose their preferred level of abstraction and to support a reduction in the amount of information displayed on a graphical interface, Triggers and Actions are hierarchical organized by functionality in two levels:

1. *High-Level.* It models *generic* triggers to be verified, or actions to be executed, and it does not include any technical detail, *nor* the type of device or service to be used to implement the desired behavior.
2. *Medium-Level.* It models *specific* triggers to be verified, or actions to be executed, and it *allows* the specification of the generic devices or services type to be used, without including any technological detail.

As an example, Figure 2 shows a partial view of the hierarchical tree that characterizes lighting-related actions. *Illuminate* and *Get Darker* actions belong to the High-Level layer, since the behaviors they define can be achieved in different ways, e.g., by turning the lights on or off, or by closing or opening a blind. If an end-user is interested in better specifying the desired operation,

she can use the Medium Level, which for lighting-related actions includes *Turn Lights On*, *Close the Blinds*, etc.

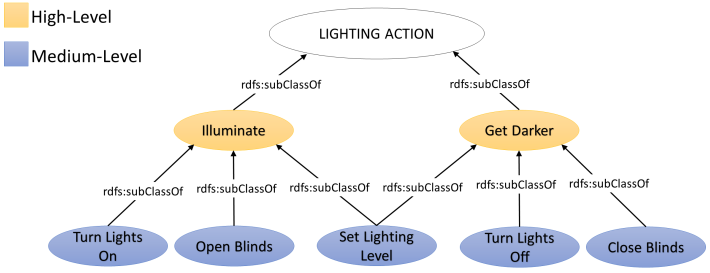


Figure 2: A partial view of the hierarchical class tree that characterizes lighting-related actions.

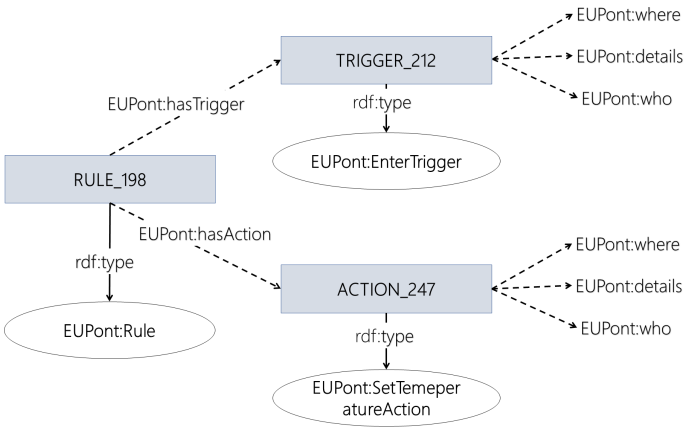


Figure 3: Graphical representation of the rule *"if I enter in a place, then set the temperature to..."* as modeled in EUPont.

To further clarify the modeling approach, Figure 3 graphically shows how the rule *"if I enter in a place, then set the temperature to..."* is modeled in EUPont. The rule has exactly one trigger of type *EnterTrigger*, and one action of type *SetTemperatureAction*. With the relationship *details*, it is possible to define a list of related details for both the Action and the Trigger, e.g., the temperature value to be set. Furthermore, through the two relationships *who* and *where*, the Trigger and the Action can be related to the involved User(s)

and Location(s), respectively, thus linking the TAP concepts to the CI layer. Such properties can be generic (e.g., any friends, or any closed space) or specific (e.g., my friend Mark, or my kitchen).

The **IoT Ecosystem (IoT) Layer** models IoT devices and services on the basis of their categories (e.g., environment systems, user devices, etc.) and final capabilities (e.g., switching capabilities, communication capabilities, etc.). For this purpose, the IoT Entity concept is specialized in various subclasses that represent different device and service types. Furthermore, each IoT Entity can offer one or more Service(s), that represent a particular capability. Service(s) may offer Command(s), i.e., actions that can be executed, and Notification(s), i.e., events that can be registered.

Figure 4 graphically shows how a real device can be modeled in EUPont. The *NEST_HOME* individual, that represent a specific Nest thermostat, offers a *HeatingService*, which allows to set a target temperature through the *NEST_SET_COMMAND*, an instance of the *SetToCommand* class. Through the two relationships *isIn* and *isOf*, the Nest thermostat can be linked to its position and to the users that are authorized to control it, respectively. In this way, the IoT layer is linked with the CI layer.

Finally, High Level Rules in the model need to be executed on real devices and services, i.e., there must be a link between the IoT layer and the TAP layer. Thanks to **semantic reasoning**, the ontology itself is able to provide such a link, by automatically mapping the defined EUPont Rules with real devices and services in the IoT Ecosystem *able* to reproduce the desired behaviors. Such a reasoning process is supported by a set of predefined *SWRL* rules. The goal of SWRL rules is to instantiate one or more *allowTo* object properties, i.e., the actual link between real devices and services and EUPont Rules. To exemplify the mechanism, Figure 5 reports the SWRL rule that links the *NEST_HOME* thermostat (Figure 4) with the Action for setting the temperature (Figure 3). Thanks to the SWRL rules, when a *SetTemperatureAction* is defined, any *IoTEntity* that offers a *HeatingService* with a *SetToCommand* is automatically considered able to reproduce the *SetTemperatureAction*, i.e., an *allowTo* is au-

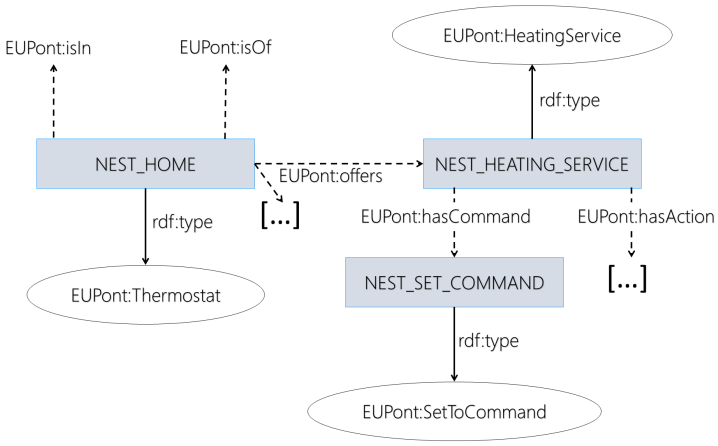


Figure 4: A device (Nest thermostat) modeled in the EUPont representation. The ovals represent classes, rectangles represent class instances (i.e., individuals), while arrows define object properties, i.e., relationships between individuals.

tomatically instantiated between *NEST_HOME* and *ACTION_247*.

```
SetTemperatureAction (?action) ^ IoTEntity(?entity) ^
offers(?entity,?service) ^ HeatingService(?service) ^
hasCommand(?service,?command) ^
SetToCommand(?command) -> allowTo(?IoTEntity, ?action)
```

Figure 5: An example of a SWRL rule defined in EUPont

Since the *SetTemperatureAction* is a Medium Level action, the same *NEST_HOME* thermostat is also automatically able to reproduce the High-Level behaviors that are parents of that action in the TAP hierarchy, e.g., *Heat Up a Place*. Such an *ability* connection, along with the information stored on the shared Contextual Information layer, can be used at run-time by EUD platforms to execute EUPont rules onto real devices and services, according to the contextual situation.

4. Architecture of an EUPont-powered Platform

To evaluate the feasibility of our approach, we designed an EUD platform that exploits EUPont for the definition and the execution of abstract rules. Such

a platform was then implemented for a smart home scenario, which serves as a working example in this section.

4.1. Platform Architecture

The architecture of the designed platform (Figure 6) can be divided in three main layers: the real *IoT devices and services*, the *Middleware layer*, and the *EUD server*.

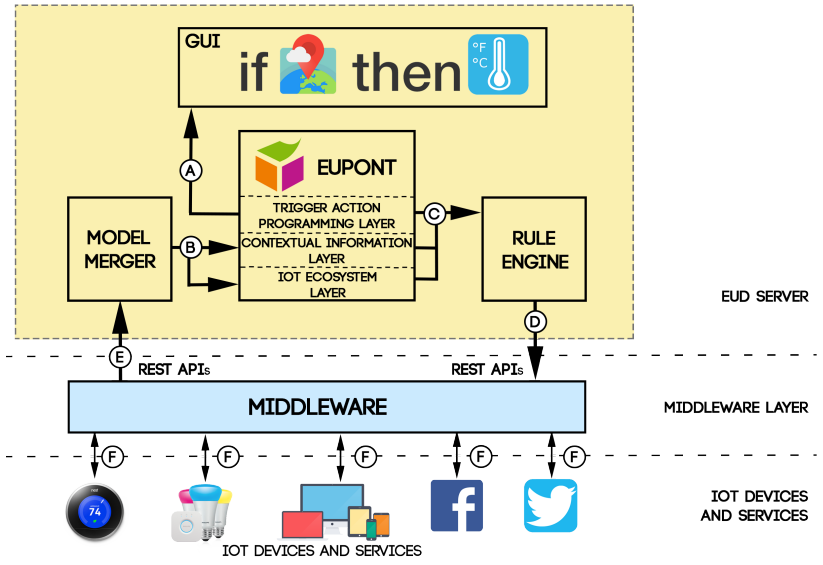


Figure 6: Overall organization of the platform that we designed for the definition and the execution of trigger-action rules in the EUPont representation

The **Middleware layer** aims at simplifying the communication between the *EUD server* and the heterogeneous IoT ecosystem, where the EUPont rules need to be executed. To communicate with real *IoT devices and services* (relationships F in Figure 6), any IoT middleware systems that exposes REST APIs can be exploited, ranging from smart home gateways to mobile applications. In the implemented smart home scenario we adopted the open source Dog [33] gateway as the Middleware layer, due to its support to multiple smart devices

and services[9]. The Dog gateway, in particular, already provides an abstraction of IoT devices by means of the DogOnt [25] ontology[10].

The **EUD server** allows the definition of EUPont rules and supports their executions. It contains four main modules: the EUPont ontology, the Graphical User Interface (GUI), the Model Merger, and the Rule Engine. The core element of this layer is the *EUPont* ontology, described in the previous section.

By interacting with the *Graphical User Interface (GUI)*, end-users can compose trigger-action rules in the EUPont representation. The GUI is loaded with triggers and actions defined in the Trigger-Action Programming layer of EUPont (relationship A in Figure 6), thus allowing the usage of the High-Level and Medium Level abstraction. Furthermore, the usage of technology-independent triggers and actions allow the GUI to reduce the amount of displayed information. Different GUIs can be implemented to exploit the EUPont representation, ranging from web-based interfaces to mobile applications. For the smart home scenario we realized a web-based GUI, similar to the one employed for the user study.

Finally, the **Model Merger** and the **Rule Engine** blocks are responsible for the communication with the real-word context (i.e, the actual IoT ecosystem). In particular, the *Model Merger* first collects real time information regarding real devices and services, e.g., their positions, their statuses, and their capabilities (relationship E in Figure 6). Then, it translates such information into OWL axioms, to continuously update the Contextual Information and IoT Ecosystem layers of EUPont (relationship B in Figure 6). Since it has to deal with domain-specific information, the implementation of the *Model Merger* depends on the real-world context in which the platform operates. In the implemented smart home scenario, the merge between EUPont and the information coming from Dog happens through a dedicated Java module that exploits the middleware's REST API. The *Rule Engine*, instead, is in charge of mapping the defined

[9] `https://dog-gateway.github.io` (last visited on October 16, 2018)

[10] `http://elite.polito.it/ontologies/dogont.owl` (last visited on October 16, 2018)

EUPont rules onto real devices and services (relationship D in Figure 6). The block is strongly supported by the ontology (relationship C in Figure 6): by reasoning on the contextual information included in EUPont and by exploiting the available SWRL rules, the Rule Engine can select real devices and services that are a) *currently available*, and b) *able* to perform the desired abstract behaviors. Also in this case, the implementation of the *Rule Engine* depends on the context in which the rules need to be executed. In the smart home scenario, we used the Dog API to directly execute actions and register events on real devices, according to the defined EUPont rules.

5. Evaluation

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 IoT applications both with the current low level representation of IFTTT and the high level representation of EUPont. The study addressed the following research questions:

- **RQ1**: Does the EUPont representation help users creating their IoT applications more effectively and efficiently compared with the low level representation?
- **RQ2**: 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 *low level* (LL) interface resembles the representation adopted by IFTTT (Figure 7), but with a limited number of channels, our *high level* (HL) interface allows users to create trigger-action rules with the EUPont representation, i.e., through the Medium and High-Level of abstraction of EUPont. For the HL interface, in particular, we used the High-Level of abstraction for triggers, and the Medium-Level of abstraction for actions. We

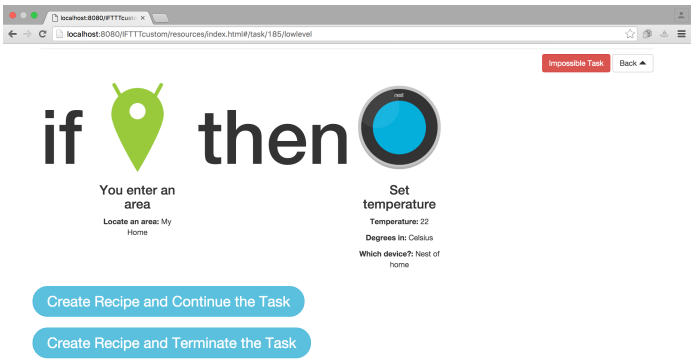


Figure 7: The study interface in the LL representation, showing the recipe "If I am approaching home, then set my Nest thermostat to 22 Celsius degree"

made this choice according to previous works in the literature, e.g., [16], that show that users are typically more abstract when referring to event, while they are more specific in defining actions. Figure 8 shows an example of the HL interface with the recipe "Set the temperature to 22 Celsius degree when I enter a place". Here, Place is the trigger channel and Temperature is the action channel.

Table 2 reports the number of channels, actions, and triggers present in both interface versions. The interfaces incorporate the *minimum* number of channels, triggers, and actions needed to complete the study. Since the EUPont representation can help minimize the amount of information presented to the users, the number of triggers and actions in the two interfaces are different.

Table 2: Channels, triggers, and actions in the interfaces

Interfaces	**Trigger Channels**	**Action Channels**	**Triggers**	**Actions**
Low Level	54	42	294	126
High Level	9	7	23	10

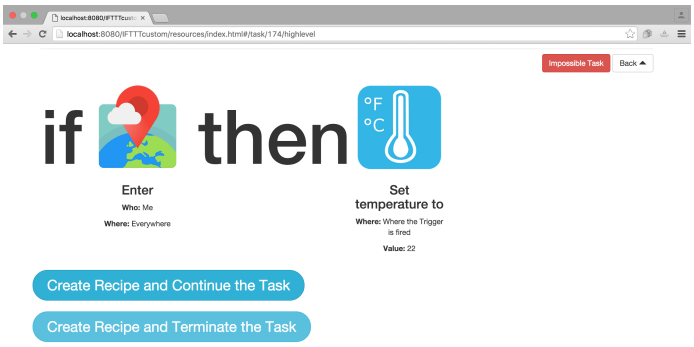


Figure 8: The study interface in the HL representation, showing the recipe "If I enter any place, then set the temperature to 22 Celsius degree"

5.1. Methodology

The study was composed of five tasks related to the creation of trigger-action rules. All the participants performed all the five tasks twice, once with the LL and once with the HL interface. We followed a two-way mixed design. We considered the used representation (LL or HL) 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 their 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.

5.1.1. 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 the IoT world. 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 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 IoT devices and services, 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 IoT devices 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).

5.1.2. Procedure

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 channel involved in the study, and we composed a trigger-action rules in both the LL and HL representations 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 representations were counterbalanced. At the end each session, we performed a final debriefing with the participant, with the aim of finding the perceived advantages and disadvantages of the two experimented representations. Task descriptions, questionnaires, debriefing questions and answers, and moderator interventions were in Italian, and translated for the purpose of this

Table 3: General Demographics

Characteristics	**Values**	**POLITO**	**UNITO**
Gender	Male	13	2
	Female	2	13
Age	20-24	10	15
	25-29	5	0
Education	Undergrad	8	15
	Grad	7	0
Programming Experience	1 - Very Low	0	13
	2 - Below Average	1	2
	3 - Average	8	0
	4 - Above Average	6	0
	5 - Very High	0	0
IoT Experience	1 - Very Low	0	9
	2 - Below Average	6	3
	3 - Average	7	2
	4 - Above Average	2	0
	5 - Very High	0	1
IFTTT Experience	1 - Very Low	10	15
	2 - Below Average	3	0
	3 - Average	2	0
	4 - Above Average	0	0
	5 - Very High	0	0

paper.

5.1.3. Tasks

We designed five tasks of the same type to be completed with both LL and HL interface. Each task consisted of two different parts: a *user scenario* and a *goal*. The *user scenario* described a generic user, her owned devices, and some of her typical activities. It did not contain any explicit reference to the general categories of the ontology. Moreover, to avoid any biases towards the high-level abstraction of EUPont, the user scenario reflected the contemporary low-level abstraction by specifying all the technologies owned by the involved users in "low-level" terms, i.e., with manufacturers and brands. The *goal* defined a specific behavior that the user wanted to obtain from her devices, and it was definable with one or more trigger-action rules. All the tasks (from T1 to T5) are reported in Appendix A. An example of a task (T1) was:

> **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.

To investigate whether the participants were aware of the the potentials and the limitations of the two representations, tasks were divided in a) solvable (completely or at least approximately) in both the representations (T1, T2,

and T4), or b) impossible to be solved in the low-level representation (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 trigger-action rules, or, in any moment, they could mark a task as impossible if they think that the rule is not completely realizable. For example, proximity of other people (e.g., T5) can be included in HL rules, but they is not supported by contemporary LL representations. Figure 8 shows the end of a rule insertion process in a task resolution, with the buttons to continue, terminate, or mark the task as impossible (top right).

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 representation:

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

In the low-level representation, 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.

5.2. Measures

Data from the study depended on two main factors (independent variables): the representation used to carry out a task (LL or HL), 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[11], b) the number of wrong triggers and c) the number of wrong 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:

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

We 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.

5.3. Quantitative Results

To answer **RQ1**, we analyzed the effect of the representation independent variable (LL or HL) over the four dependent variables. We considered wrong triggers, wrong actions, and back number as measures able to indicate how a

[11] except for think-aloud tasks

representation *effectively* helps and guides users in defining their IoT applications. The duration measure was indeed used to investigate the *efficiency* of the rule definition process. Table 4 reports the means of the four analyzed measures in both representations.

Table 4: Means and standard deviations of the four normalized dependent variables investigated in the study

	LL	**HL**
Wrong Triggers	0.279 ± 0.028	0.120 ± 0.025
Wrong Actions	0.203 ± 0.028	0.038 ± 0.010
Back Number	0.971 ± 0.150	0.588 ± 0.117
Duration	$39.647s \pm 2.600s$	$25.054s \pm 1.948s$

5.3.1. Effectiveness Results

To investigate whether the HL representation allows users to create IoT applications more effectively, we conducted three different two-way mixed ANOVA in SPSS with a post-hoc analysis with Bonferroni correction. We considered wrong triggers, wrong actions, and the number of back as the dependent variables; the used representation 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 three analysis.

Wrong Triggers. 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 composed with different representations significantly differ ($F(1, 28) = 19.14$, $p < .05$). Figure 9 indicates that participants composed HL rules with less errors during the definition of triggers. The means of the wrong triggers, in fact, were lower with the HL than with the LL representation (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

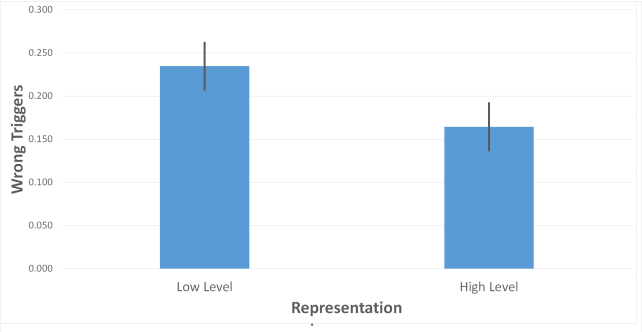


Figure 9: The means for the main effect of the representation on the wrong triggers

a significant interaction between the used representation and the programming experience in terms of wrong triggers ($F(1, 28) = 8.348 \times 10^{-6}$, $p > .05$).

Wrong Actions. Also for the wrong actions variable, we found that there was a significant main effect of the used representation ($F(1, 28) = 35.837$, $p < .05$). As for the triggers, by ignoring the programming experience, the number of errors in the definition of the actions for rules composed with different representations significantly differs. Figure 10 indicates that participants made less errors during the definition of actions in HL rules, as the means of the wrong actions variable were higher with the LL than with the HL representation (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 ($p < .05$).

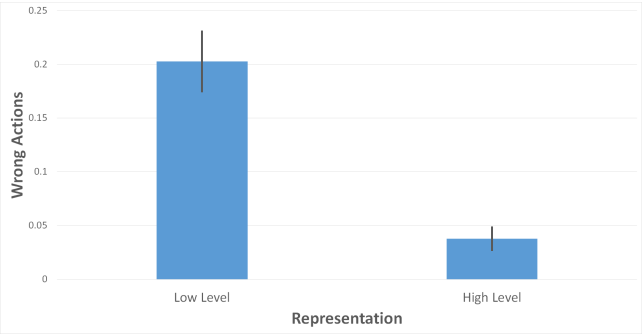


Figure 10: The means for the main effect of the representation on the wrong actions

The programming experience level of the participants did not significantly influence the wrong actions variable ($F(1, 28) = 0.001$, $p > .05$) nor there was a significant interaction between the representation and the programming experience in terms of wrong actions ($F(1, 28) = 0.343$, $p > .05$).

Back Number. Finally, we found that there was not a significant main effect of the used representation on the back number variable ($F(1, 28) = 4.152$, $p > .05$). However, the back number was lower with the HL representation (0.588 ± 0.117 vs. 0.971 ± 0.150), as shown in Figure 11.

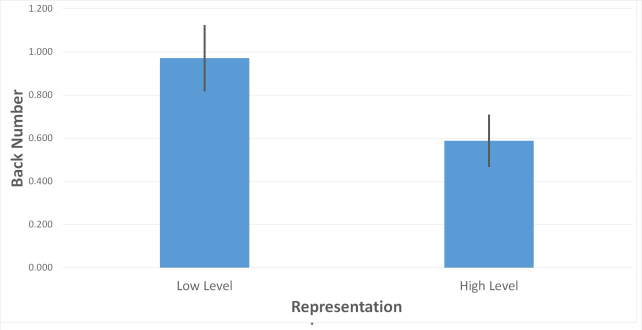


Figure 11: The means for the main effect of the representation on the back number

5.3.2. Efficiency Results

To investigate whether the HL representation allows users to create IoT applications more efficiently, we conducted another two-way mixed ANOVA. This time, we considered the duration as the dependent variables. The Mauchly's sphericity test was satisfied for the independent variable (i.e., the used representation). We found that the average time needed for composing rules was significantly different for the two representations ($F(1, 28) = 25.402$, $p < .05$), independently from the programming experience. Users composed rules with the HL representation faster than with the LL representation. In fact, the means of the duration variable were lower for the HL than for the LL representation ($25.054s \pm 1.948s$ 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 representation and the programming experience in terms of duration ($F(1, 28) = 0.354$, $p > .05$).

5.4. Qualitative Results

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 (**RQ2**). 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.

5.4.1. Understanding Low Level Limits

The limits of the LL 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 is impossible to define with the LL representation, 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 with the low level functionality.*" All of these 15 participants correctly stated that the task was impossible to complete. Other 5 participants defined the event anyway, but they explicitly acknowledged to be 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.*"

5.4.2. Avoiding Mistakes and Confusion

From the analysis of the thoughts and behaviors of the participants, it emerges that the HL representation helped users avoid mistakes in the rules composition. During the solution of a task with the LL 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 low level representation, 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 with many different technologies, as in the task reported in subsection 5.1.3. In this case, 11 participants forgot to replicate the same rule for all the different devices and rooms, or they wrongly defined some triggers or actions details (e.g., the specific light type). With more general attributes, the HL representation seemed to mitigate this problem, since 26 participants completed the tasks without any difficulty and correctly. The video recordings analysis shows that users seemed to be more confused in defining IoT applications with the LL representation. 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 HL representation without changing idea (48.3%).

5.4.3. Advantages and Disadvantages

By analyzing the data from the debriefings, we coded the feedback given by the participants about the *low level representation* in some disadvantages and advantages. First of all, the participants recognized the *adaptation* and *abstraction* issues identified in Section 1. They said that a user has to know in advance the devices she owns to define her IoT applications. This disadvantage was especially cited by participants without programming experience. Starting from this fact, tasks in the LL representation required various (complex) rules, and the required time to define IoT applications was higher than the time re-

quired with the HL representation. The majority of the participants agreed that generic concepts were impossible to define with the LL representation. Other users were disappointed about the large number of channels, often perceived as not necessary. Not surprisingly, the specific nature of the LL representation 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).

Similarly, participants found advantages and disadvantages for the *high level representation*. The first disadvantage that emerges matches with some advantages of the low level representation. In fact, participants said that sometimes the HL interface seemed to be too generic, and it did not offer the possibility to manage the details of the involved services. For this reason, a user does not have full control of what happens, and she must trust the system. Two participants were worried about privacy and security issues for a system based on the HL representation. From the participant answers we retrieved consistent advantages of the high level representation as well. Participants were satisfied about the generic abstraction of channels, actions, and triggers. The HL representation allowed the definition of generic behaviors and the composition of fewer rules. Furthermore, the representation was simple, and easy to use and understand, especially for non-programmers. Finally, defining IoT applications with the HL representation required less time.

6. Discussion

In this section, we discuss the results of our user study by summarizing the resulting insights along different themes.

Reducing Displayed Information. In the upcoming IoT ecosystem, devices and services will not always be knowable a priori and the complexity of the entire IoT ecosystem will continuously increase. As some participants in our study found, a low-level representation risks to generate user interfaces that are cluttered and with too much information. For realizing the low-level interface, in

fact, we needed to provide 54 trigger channels and 42 action channels in the low-level (Table 2). The high-level interface, instead, adopted 9 trigger channels and 7 action channels, only, to provide users with equivalent functionality. By presenting a lower number of channels, the high-level representation powered by EUPont allowed participant to compose IoT applications in less time, with less errors, and with more guidance towards the choices they had. As a result, using EUPont supported the reduction of the amount of displayed information helped and guided participants in the composition of trigger-action applications. Such a result might produce benefits even for interfaces based on low-level representations, e.g., by showing to the user only her owned devices and services.

Programming by Functionality. The *leitmotif* off all our work on End-User Development in the IoT 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. As suggested by Ur et al. [16], in fact, 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 a rule aims to define. For such a reason, we designed EUPont 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 medium/high-level trigger or action, e.g., all the lamps offered the same *turn the light 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, helps users defining their IoT applications in terms of effectiveness and efficiency.

Custom Level of Abstraction. While a higher level of abstraction presents several benefits, some participants of the study provided interesting issues related to the EUPont representation. When using the High-Level of abstraction of EUPont for defining triggers, in fact, the interface sometimes appeared to be too generic and did not offer the possibility to manage particular details of involved devices and services. Even if 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 IoT applications (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 a more precise control. However, they desired not to have the same amount of details as in the low-level representation. Therefore it could be useful to provide users with 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 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. [2], in which the authors explored the trigger-action paradigm in the smart home scenario. In particular, consistently with other related works [5, 34], they found that participants tended not to mention sensors directly, and they discovered that many participants express triggers one level of abstraction higher.

Context-dependency. During the 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 wrongly defined some specific trigger or action details. Participants explicitly said that to reach a goal they had to insert several rules in the low-level representation, and the required time to define IoT applications was high. Furthermore, by analyzing the experiment videos and interviews, we found

that often users would like to reuse the same 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 [4].

Trustfulness, Security, and Privacy. Some participants in the user study highlighted that they should have a profound *trust* in a system that adopted the presented high-level representation. 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. When adopting a high-level of representation such as EUPont, trustfulness, privacy, and security issues must be taken in serious consideration. This may include warning mechanisms in the interface to alert users about possible dangerous rules, and debug features to help people simulate and foresee rule behavior under different conditions [30]. Debug could be even more useful in case of abstract trigger-action rules: based on the context, showing on which real devices and services a high-level rules is mapped onto could increase the system trustfulness.

7. Conclusions and Future Work

In this paper, we explored a new way of personalizing IoT devices and services. The "low level" representation adopted by contemporary Task Automation tools (e.g., IFTTT) is, in fact, not suitable to overcome the issues brought on by the steady growth of the IoT ecosystem. The fact that devices and services with similar capabilities but different brands are treated as separate entities (*abstraction* issue), and that contemporary solutions only work with well-known devices and services (*adaptation* issue) are two evident examples. This need guided us in the formal definition of EUPont, an ontological high-level representation for End-User Development in the IoT: we chose to exploit semantic technologies and we included in the model features to support the actual execution of the IoT applications. To evaluate the feasibility of the

approach and to actually execute the abstract rules provided by EUPont, we designed and implemented an EUD platform. In the platform, the EUPont model allows the definition of IoT applications, and the selection of *currently available* real devices and services *able* to reproduce the defined abstract behaviors.

Thanks to a user study with 30 participants, we also successfully demonstrate that the EUPont representation allows end-users to reduce errors and time needed to compose their IoT applications, and introduces numerous benefits in terms of understandability and ease of use. Of course, we are aware that there are many aspects to be better explored, especially for the execution of EUPont rules. For example, high level behaviors such as *"Illuminate a place"* or *"Send a message"* could be potentially reproduced in different ways, on the basis of the current context: which is the "best" solution for a user? Finally, it is worth noticing that trustfulness, security, and privacy issues become even more important with a high level representation. If we imagine IoT applications such as *"if I enter a closed space, then set the temperature to 20 Celsius degree"*, which IoT devices and services 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? These open questions will guide our future works.

Appendix A. Evaluation Tasks

Here, we report the 5 tasks used in the evaluation. T1 and T5 were carried our with the think-aloud protocol.

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 he 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 update 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 take 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.

References

[1] D. Evans, The Internet of Things: How the Next Evolution of the Internet Is Changing Everything, Tech. rep., Cisco Internet Business Solutions Group (2011).

[2] B. Ur, E. McManus, M. Pak Yong Ho, M. L. Littman, Practical trigger-action programming in the smart home, in: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '14, ACM, New York, NY, USA, 2014, pp. 803–812. doi:10.1145/2556288.2557420.

[3] J. Saunders, D. S. Syrdal, K. L. Koay, N. Burke, K. Dautenhahn, "Teach Me - Show Me" - End-User Personalization of a Smart Home and Companion Robot, IEEE Transactions on Human-Machine Systems 46 (1) (2016) 27–40. doi:10.1109/THMS.2015.2445105.

[4] G. Ghiani, M. Manca, F. Paternò, C. Santoro, Personalization of context-dependent applications through trigger-action rules, ACM Transactions on Computer-Human Interaction (TOCHI) 24 (2) (2017) 14:1–14:33. doi:10.1145/3057861.

[5] A. K. Dey, T. Sohn, S. Streng, J. Kodama, iCAP: Interactive prototyping of context-aware applications, in: Proceedings of the 4th International Con-

ference on Pervasive Computing, PERVASIVE'06, Springer-Verlag, Berlin, Heidelberg, 2006, pp. 254–271. doi:10.1007/11748625_16.

[6] J. Lee, L. Garduño, E. Walker, W. Burleson, A tangible programming tool for creation of context-aware applications, in: Proceedings of the 2013 ACM International Joint Conference on Pervasive and Ubiquitous Computing, UbiComp '13, ACM, New York, NY, USA, 2013, pp. 391–400. doi:10.1145/2493432.2493483.

[7] M. Coronado, C. A. Iglesias, Task automation services: Automation for the masses, IEEE Internet Computing 20 (1) (2016) 52–58. doi:10.1109/MIC.2015.73.

[8] A. Zaslavsky, P. P. Jayaraman, Discovery in the internet of things: The internet of things (ubiquity symposium), Ubiquity 2015 (October) (2015) 2:1–2:10. doi:10.1145/2822529.

[9] B. R. Barricelli, S. Valtolina, End-User Development: 5th International Symposium, IS-EUD 2015, Madrid, Spain, May 26-29, 2015. Proceedings, Springer International Publishing, Cham, Germany, 2015, Ch. Designing for End-User Development in the Internet of Things, pp. 9–24. doi:10.1007/978-3-319-18425-8_2.

[10] F. Corno, L. De Russis, A. Monge Roffarello, A Semantic Web Approach to Simplifying Trigger-Action Programming in the IoT, IEEE Computer 50 (11) (2017) 18–24. doi:10.1109/MC.2017.4041355.

[11] H. Lieberman, F. Paternò, M. Klann, V. Wulf, End User Development, Springer Netherlands, Dordrecht, Netherlands, 2006, Ch. End-User Development: An Emerging Paradigm, pp. 1–8. doi:10.1007/1-4020-5386-X_1.

[12] D. Munjin, User Empowerment in the Internet of Things, Ph.D. thesis, Université de Genève (May 2013).
URL http://archive-ouverte.unige.ch/unige:28951

[13] I. P. Cvijikj, F. Michahelles, Architecting the Internet of Things, Springer Berlin Heidelberg, Berlin, Heidelberg, 2011, Ch. The Toolkit Approach for End-user Participation in the Internet of Things, pp. 65–96. doi:10.1007/978-3-642-19157-2_4.

[14] J. Danado, F. Paternò, Puzzle: A mobile application development environment using a jigsaw metaphor, Journal of Visual Languages & Computing 25 (4) (2014) 297–315. doi:10.1016/j.jvlc.2014.03.005.

[15] D. Le-Phuoc, A. Polleres, M. Hauswirth, G. Tummarello, C. Morbidoni, Rapid prototyping of semantic mash-ups through semantic web pipes, in: Proceedings of the 18th International Conference on World Wide Web, WWW '09, ACM, New York, NY, USA, 2009, pp. 581–590. doi:10.1145/1526709.1526788.

[16] B. Ur, M. Pak Yong Ho, S. Brawner, J. Lee, S. Mennicken, N. Picard, D. Schulze, M. 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, ACM, New York, NY, USA, 2016, pp. 3227–3231. doi:10.1145/2858036.2858556.

[17] J. Huang, M. 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, ACM, New York, NY, USA, 2015, pp. 215–225. doi:10.1145/2750858.2805830.

[18] L. Atzori, A. Iera, G. Morabito, The Internet of Things: A Survey, Computer Networks: The International Journal of Computer and Telecommunications Networking 54 (2010) 2787–2805. doi:10.1016/j.comnet.2010.05.010.

[19] P. Barnaghi, W. Wang, C. Henson, K. Taylor, Semantics for the internet of things: Early progress and back to the future, International Journal on

Semantic Web and Information Systems 8 (1) (2012) 1–21. doi:10.4018/jswis.2012010101.

[20] M. Compton, P. Barnaghi, L. Bermudez, R. Garcìa-Castro, O. Corcho, S. Cox, J. Graybeal, M. Hauswirth, C. Henson, A. Herzog, V. Huang, K. Janowicz, W. D. Kelsey, D. Le Phuoc, L. Lefort, M. Leggieri, H. Neuhaus, A. Nikolov, K. Page, A. Passant, A. Sheth, K. Taylor, The SSN ontology of the W3C semantic sensor network incubator group, Web Semantics: Science, Services and Agents on the World Wide Web 17 (2012) 25–32. doi:10.1016/j.websem.2012.05.003.

[21] M. Botts, G. Percivall, C. Reed, J. Davidson, OGC sensor web enablement: Overview and high level architecture, in: S. Nittel, A. Labrinidis, A. Stefanidis (Eds.), GeoSensor Networks, Springer-Verlag, 2008, pp. 175–190. doi:10.1007/978-3-540-79996-2_10.

[22] M. Bermudez-Edo, T. Elsaleh, P. Barnaghi, K. Taylor, Iot-lite: A lightweight semantic model for the internet of things, in: Proceedings of 13th International Conference on Ubiquitous Intelligence and Computing, 2016, (in press).

[23] S. De, T. Elsaleh, P. Barnaghi, S. Meissner, An internet of things platform for real-world and digital objects, Scalable Computing: Practice and Experience 13 (1) (2012) 45–57.
URL http://epubs.surrey.ac.uk/531903/

[24] W. Wang, S. De, G. Cassar, K. Moessner, Knowledge representation in the internet of things: Semantic modelling and its applications, Automatika - Journal for Control, Measurement, Electronics, Computing and Communications 54 (4) (2013) 388–400.
URL http://epubs.surrey.ac.uk/794745/

[25] D. Bonino, L. De Russis, DogOnt as a viable seed for semantic modeling of AEC/FM, Semantic Web 9 (6) (2018) 763–780. doi:10.3233/SW-180295.

[26] J. Kramer, O. Hazzan, The role of abstraction in software engineering, SIGSOFT Softw. Eng. Notes 31 (6) (2006) 38–39. doi:10.1145/1218776.1226833.

[27] C. Ardito, P. Buono, G. Desolda, M. Matera, From smart objects to smart experiences: An end-user development approach, International Journal of Human-Computer Studies 114 (2018) 51 – 68, advanced User Interfaces for Cultural Heritage. doi:https://doi.org/10.1016/j.ijhcs.2017.12.002.

[28] D. Salber, A. K. Dey, G. D. Abowd, The context toolkit: Aiding the development of context-enabled applications, in: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '99, ACM, New York, NY, USA, 1999, pp. 434–441. doi:10.1145/302979.303126.

[29] T. Gu, H. K. Pung, D. Q. Zhang, A service-oriented middleware for building context-aware services, Journal of Network and Computer Applications 28 (1) (2005) 1 – 18. doi:10.1016/j.jnca.2004.06.002.

[30] G. Desolda, C. Ardito, M. Matera, Empowering end users to customize their smart environments: Model, composition paradigms, and domain-specific tools, ACM Transaction on Computer-Human Interaction (TOCHI) 24 (2) (2017) 12:1–12:52. doi:10.1145/3057859.

[31] P. Hu, J. Indulska, R. Robinson, An autonomic context management system for pervasive computing, in: 2008 Sixth Annual IEEE International Conference on Pervasive Computing and Communications (PerCom), 2008, pp. 213–223. doi:10.1109/PERCOM.2008.56.

[32] D. Bonino, E. Castellina, F. Corno, The DOG gateway: enabling ontology-based intelligent domotic environments, IEEE Transactions on Consumer Electronics 54 (4) (2008) 1656–1664. doi:10.1109/TCE.2008.4711217.

[33] D. Bonino, F. Corno, L. De Russis, A semantics-rich information technology

architecture for smart buildings, Buildings 4 (4) (2014) 880–910. doi:10.3390/buildings4040880.

[34] K. N. Truong, E. M. Huang, G. D. Abowd, UbiComp 2004: Ubiquitous Computing: 6th International Conference, Nottingham, UK, September 7-10, 2004. Proceedings, Springer Berlin Heidelberg, Berlin, Heidelberg, 2004, Ch. CAMP: A Magnetic Poetry Interface for End-User Programming of Capture Applications for the Home, pp. 143–160. doi:10.1007/978-3-540-30119-6_9.