Engineering Task Automation Systems for Domain Specificity

Carmelo Ardito[1], Giuseppe Desolda[1], Maristella Matera[2]

[1]Dipartimento di Informatica, Università degli Studi di Bari Aldo Moro
Via Orabona, 4 – 70125 – Bari, Italy
{name.surname}@uniba.it
[2]Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano
Piazza Leonardo da Vinci, 32 – 20134 – Milano, Italy
maristella.matera@polimi.it

Abstract. Domain specificity is largely recognized as a means to foster the adoption of systems by specific communities of non-technical users. This paper presents an architecture for the development of Task Automation Systems that can be customized in specific domains. It is one of the results of a human-centred design process we performed to support non-technical people to program the behaviour of smart objects by defining event-condition-action (ECA) rules. We illustrate the main modules of the proposed architecture, also describing how it supports the creation of ECA rules constrained by means of temporal and spatial conditions. Finally, we report on the development of a Task-Automation System customized by developing and comparing three different composition paradigms.

Keywords: Task Automation Systems, Internet of Thing, End-User Development, Domain Specificity

1 Introduction

In the last years, the spreading of low-cost technologies that integrates sensors and actuators has made easier building the so-called *smart objects*. A smart object is an electronic device connected to the Internet, which embeds sensors to feel the environment and/or actuators to communicate with the environment [5]. The proliferation of such devices led to the Internet of Things (IoT), a novel paradigm where the Internet is connected to the physical world via ubiquitous sensors[1]. The IoT is breeding grounds for different research areas since several challenges need to be addressed, such as those related to energy consumption, communication protocols, programming languages and end-user development (EUD) [14, 23]. Many efforts are being devoted to improve technological features. Little attention has been instead dedicated to social and practical aspects: therefore, despite all the advances in the IoT field, end users still encounter difficulties when they try to make sense of such technology. The research community

[1] http://www.rfidjournal.com/articles/view?4986

agrees on the fact that the opportunities offered by IoT can be amplified if high-level abstractions and adequate interaction paradigms are devised to enable also non-programmers to customize and synchronize the behaviour of smart objects [32].

In line with whit this claim, Task Automation Systems (TAS) become a popular solution to support non-technical users, i.e., people without skills in computer programming, to synchronize smart objects by exploiting visual mechanisms [24]. Despite a wide availability of TASs, their graphical notations often do not match the mental model of most users [33]. In addition, TASs are typically conceived as general purpose systems but their generality often implies a scarce adoption by specific communities of end users [12].

This paper proposes an architecture that fosters the development of TASs that are customizable with respect to varying users and usage domains. The customization mainly consists in developing a specific User Interface (UI) that "speaks the language of the user", i.e., that proposes terminology, concepts, rules, and conventions the user is comfortable with. In addition, the architecture addresses the smart objects synchronization by means of event-condition-action (ECA) rules. Such rules are based on a model, called *5W*, which defines some specification constructs (*Which, What, When, Where, Why*) to build rules coupling multiple events and conditions exposed by smart objects, and for defining temporal and spatial constraints on rule activation and actions execution. This model meets the mental model of the users, who can easily describe the ingredients of the ECA rules following the 5W simple questions. Starting from the proposed architecture, this paper briefly illustrates the development of a TAS called EFESTO and its customization through the development of three different UIs.

The paper is organized as follows. Section 2 illustrates the architecture that drives the development of Task Automation Systems, which can be customized by developing proper UIs that satisfy varying users and usage domains. Section 3 describes the implementation of the EFESTO platform and its customization with three UIs implementing different composition paradigms. Section 4 reports related works. Finally, Section 5 concludes the paper also outlining our future work.

2 Domain Specificity in Task Automation Systems: a Platform Architecture

In this section, we illustrate an architecture that facilitates the development of Task Automation Systems that are customizable with respect to varying users and usage domains. The architecture design was driven by the need to develop a general TAS that can be easily customized by adapting in the interaction layer terminology, concepts, rules, and conventions the user is comfortable with [2], thus facilitating its adoption in different domains. The proposed architecture features a decoupling of the interaction layer from the other platform modules. Software design patterns, first of all the MVC (Model-View-Controller), already addressed this separation of concerns. In our work, however, the emphasis is not on programming practices to facilitate the development and maintenance of an interactive system; rather we want to stress the possibility to adapt easily the composition paradigm offered by the TAS, to comply with domain-

specific requirements. It is indeed important to restrict the TAS to a well-defined domain the user is comfortable with. That is, it is important to develop a general TAS that can be, however, easily customized as far as the provided composition metaphor is concerned [2].

The resulting TASs allow people to exploit different composition paradigms to program the behaviour of smart objects by defining ECA rules whose events and actions are defined in term of: *Which* is the object, *What* event triggers the rule (What action has to be activated), and *When* and *Where* the event/action has to happen [17]. This characterization of rule events and actions is inspired by the 5W model typically adopted in journalism to describe a fact.

2.1 Platform organization

The architecture inherits some modules for service invocation and management already developed in the EFESTO mashup framework [16]. The focus of the new architecture is however on the *Rule Engine*. As reported in Figure 1, the architecture is organized in three layers, each one managing a separate aspect.

The *Interaction Layer* refers to the system client that manages the UI through which the users can create ECA rules. In addition, it implements two modules, the *Service Builder* and the *Rule Generator*. The first one is in charge of materializing in the UI the list of attributes of registered services, as resulting from the *Service Descriptor* repository. Thus, it is invoked each time users need to add an event or an action to the rule. The UI layer is in principle agnostic to the registered services; to build the visualization of available services, the Service Builder requests to the *Service Engine* the JSON file containing the list of available services, each of them described by attributes like name, events, actions and thumbnail URL.

The *Rule Generator* is an interpreter that translates the user visual actions for rule creation into a JSON specification that describes the rule in terms of events, actions, logical operators and spatial and temporal constraints (see Figure 2b).

At the server side, the *Logic Layer* manages rules and services by means of respectively the *Rule Engine* and the *Service Engine* modules. The first one receives the rule JSON file (Figure 2b) from the client (from the *Rule Generator* module) and instantiates the rule object based on a publish-subscribe, event-action model [10, 11]. This model is natively managed and handled by a Java Spring class[2] for tasks scheduling. Each rule object is characterized by a set of *Publisher* services, each of them associated with an event that can be complemented with temporal and spatial constraints, and by a set of *Subscriber* services, each of them associated to an action that can be complemented with temporal and spatial constraints. Moreover, details about the logical operators used among events or actions are stored in the rule object.

[2] ThreadPoolTaskScheduler (http://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/scheduling/concurrent/ThreadPoolTaskScheduler.html)

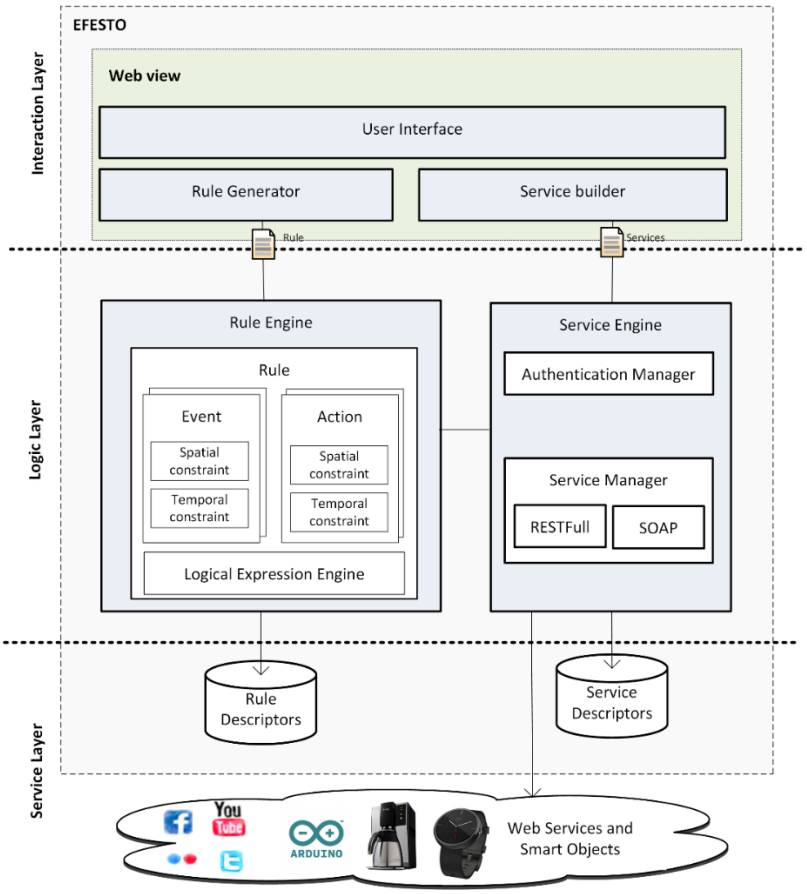

Figure 1. Overall organization of the platform architecture and structure of the rule engine.

The Rule Engine acts as an event bus that mediates the communication between the different components. Components are decoupled: they do not need to be explicitly aware of each other or be blocked waiting for events from other components. Depending on the nature of the service, the Rule Engine can work as active or passive component. In the first case, it checks every N minutes if the publisher events are triggered (all of them or just one of them depending on the logical operator, respectively AND or OR). This check is performed by a *listener* associated to the rule. In the second case, it is notified by the service when an event is triggered. In both cases, if the events are triggered, the Rule Engine controls if there are temporal and spatial constraints on the events and, in case, if they are satisfied. If the events meet all the conditions, the Rule Engine runs all the subscribed actions associated with the rule or schedules the action execution according to the when constraint.

The *Service Layer* is located at the server side and stores service and rule descriptors by using JSON files. A *service descriptor* contains all the information useful to query an API and contributes to decouple the registered services from the rest of the platform. It is created when a new object is added into the platform. Different technology (e.g.,

RESTful) can be easily accommodated as the EFESTO service layer [16] is structured so that different types of adapters can be plugged in to manage the access to different API technologies. Alternatively, without developing further adapters, it is possible to adopt a dedicated middleware, as for example Azure IoT Suite[3], to mediate the access to additional service technologies [22]. The resulting platform is indeed open and each layer can be also implemented by external services.

A b

Figure 2. a) Service descriptor of the bracelet smart object; b) JSON descriptor of a rule with 2 causes and 2 actions and

An example of service descriptor is provided in Figure 2a. It is divided into two main sections: header and body. The attributes *name* and *url* in the header specify respectively the service name and the API documentation URL. The body section includes a set of attributes (*appID*, *appSecret*, *restUri*, *redirectUri*, *tokenExpiredCode*, *authentication*) that the Service Engine uses to invoke the API. Moreover, the *functions* JSON array contains a list of *events* and *actions*, each of them characterized by the attributes

[3] https://www.microsoft.com/en/server-cloud/internet-of-things/azure-iot-suite.aspx

type, *name*, *path*, *method* and *response*, which are respectively the type of function (event or action), the event/action name displayed to the users in the UI, the event/action path chained to the *restUri* URL to invoke the event/action, the type of API call (e.g. GET, POST) and the provider response format (e.g. JSON, XML).

3 Development and Customization of a Task Automation System

In this section we describe a TAS called EFESTO-5W we developed according to the proposed architecture. The separation of the UI layer from the other two layers allowed us to customize EFESTO-5W by proposing three different composition paradigms. Each paradigm gives the name to the customized version of the platform, i.e., EFESTO-Free, EFESTO-Wizard and EFESTO-Wired, abbreviated to *E-Free*, *E-Wizard* and *E-Wired*, respectively. Lack of space prevents us to report further information about the prototype design and evaluation. Interested readers can refer to [17] for details about the design process.

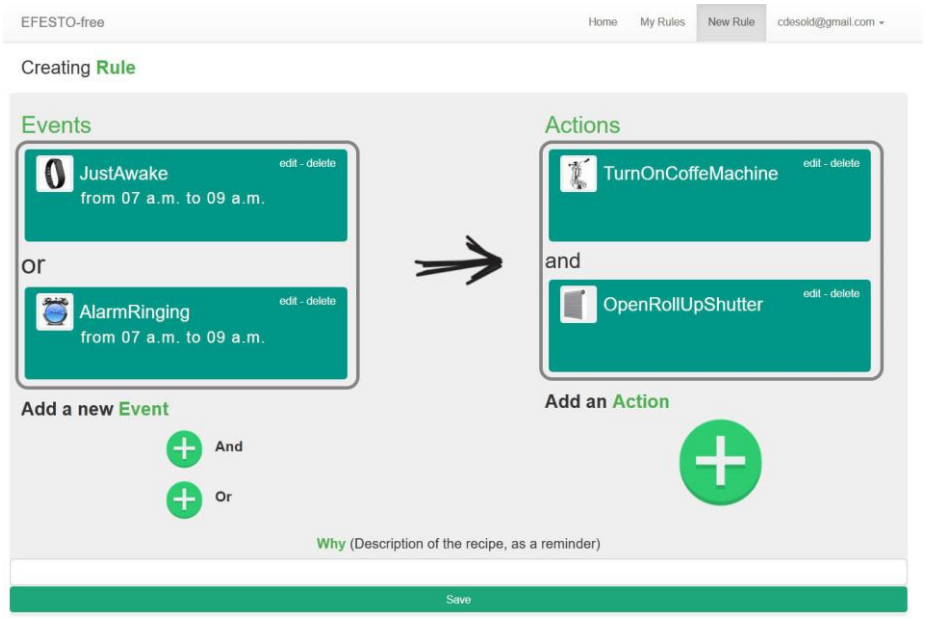

Figure 3. E-Free: example of rule including two events and two actions.

E-Free and E-Wizard propose two similar paradigms. In both the prototypes, as shown in Figure 3, the main screen where the rule is created presents two main sides, the left one to add the events and the right one to add the actions. To add an event users have to click on the + button in the Events area, thus activating a wizard procedure that assists them in defining *Which* is the service to be monitored for detecting the triggering event, *What* service event has to be monitored and *When* and *Where* the event has to be

triggered. Similarly, they can add an action by clicking the + button in the Action area to activate the wizard steps to define *Which* service will execute the action, *What* action the service has to perform and *When* and *Where* the action can be performed.

The main difference between E-Free and E-Wizard is that in E-Wizard, before to access to the main screen, users are compelled to follow a wizard procedure to create a "basic rule" composed of one event and one action. Then, they can add further events and actions exploiting the main screen reported in Figure 3. On the contrary, in E-Free the rule creation starts from the main screen and here users may either define first all the events and then the actions, or define first a basic rule including one event and one action and later include new events and new actions. Events and actions can be added or removed at any time.

E-Wired implements an interaction paradigm based on the graph metaphor: nodes represent smart objects involved in a rule, while directed edges, i.e. arrows, represent cause-effect relationships between them. As reported in Figure 4, the E-Wired UI has two main areas. The sidebar on the left provides the list of all the available smart objects and Web services: Web services are light-yellow, while smart objects are light-green. In the workspace area, users build the rule. They first have to select one of the services in the left sidebar, which is added to the workspace and represented as a box augmented with two small circles, light-blue and purple, which represent the connection points for the arrows representing cause-effect relationships. As soon as the arrow is drawn, two pop-up windows in sequence allow the user to specify the parameters of the Event and of the Action in terms of *What*, *When*, and *Where*. A "Create Rule" button in the second pop-up window permits to save the rule, also specifying *Why*, i.e., a title shortly describing the rule.

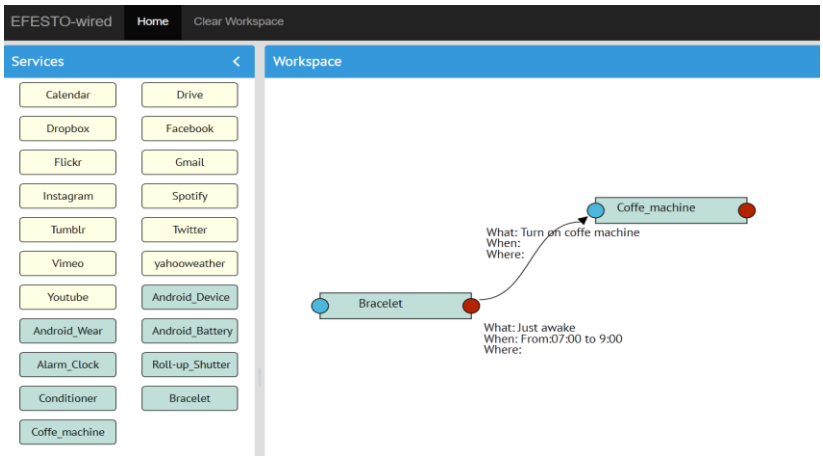

Figure 4. E-Wired: example of rule including one event and one action.

A comparative experiment among the three EFESTO-5W prototypes revealed that the composition paradigm implemented in E-Free outperforms the ones implemented in E-Wizard and E-Wired, both in term of user performances and satisfaction [17]. Starting from E-Free, we recently customized its UI exploring three visual composition

techniques to specify logical expressions in ECA rules [18]. The first technique proposes abstraction mechanisms to combine rule events by means of AND/OR logical operators, as well as to group set of conjunctive/disjunctive events, also recursively. The second technique constraints the creation of logic expressions taking into account a principle of the mental model theory [21] saying that people find easier the conceptualization of logical statements as a disjunction of conjunctions (Disjunctive Normal Form - DNF). The third technique is the opposite of DNF, since it allows the combination of rule events as a *conjunction of disjunction* (Conjunctive Normal Form - CNF). We are currently investigating pro and cons of each of these techniques in creating ECA rules.

4 Related Work

To bring close end users' desire to customize smart object behavior and the intrinsic complexity of programming languages, different solutions are emerging today. Since a smart object is remotely available as a Web service, in many cases such solutions are getting inspiration from the mashup research area. Mashup tools are Web platforms that permit to access and compose heterogeneous resources, including Web services, by exploiting visual mechanisms [15]. Starting from the mashup approaches, *task automation platforms* [13] have been proposed as means for synchronizing services and smart objects. Such tools support users in the automation of their processes by establishing channels among smart objects (e.g., each time a user enters into his home, the Wi-Fi router switch on). A popular task automation platform is IFTTT (IF This Then That): it provides wizard mechanisms for creating automation rules, called *recipes*, to throw an action on a service when an event is triggered by another service [20]. For instance, when an intrusion is detected by the home alarm system, the Smartwatch shows a notification to the user.

The wizard paradigm fits very well the mental model of non-technical end users [3], and this is the reason why it is widely exploited also by other task automation tools. An example is *elastic.io*, a tool to create rule expressing data-flow chains [1]. It is more devoted to business aspects and offers the possibility to integrate custom services. Another example is *Zapier*, whose main features are *i)* the possibility to create rules with multiple events and actions and *ii)* the use of *filters* on the triggering events to control rules activation [35]. Task automation tools implementing wizard approaches are also available as mobile apps. *Atooma* is one of the most popular; it allows the creation of rules with multiple events and actions, which put into communication device functions, Web services and smart objects [4]. A recent work demonstrated that, even if Atooma supports the creation of very expressive rules, the wizard approach guarantees similar performances between IFTTT (the mobile version) and Atooma with reference to time and accuracy [9]. Similarly to Atooma, tools like *AutomateIt* and *Tasker* support the creation of rules, but they simply enable the composition of apps and functions available on mobile devices [6, 30].

Besides the wizard-based task automation tools, there are other different composition paradigms. For example, the graph metaphor is used to represent a Web service as

a node and connections among Web services as "wires". Users can define object communication/behavior by graphically sketching the wires among the objects. A popular tool implementing the wired paradigm is *Node-RED* [31]. Besides offering a set of predefined services, it allows users to register personal smart objects by invoking their RESTful interfaces. In addition, Node-RED supports the creation of complex automation rules characterized by: *i)* multiple services that trigger events and multiple services that react by performing actions; *ii)* special nodes, used for example to control the communication flow among services by means of custom JavaScript code; *iii)* debug function to simulate and check the rules under creation. However, such features often require technical skills and thus they are not adequate for non-technical people [25, 26, 34]. The wire paradigm is implemented by tools typically devoted to more technical users, for example by *Bip.io* [8] and *Spacebrew* [29].

A completely different paradigm is implemented in *Zipato*, a platform specific for smart objects in domotic systems [36]. The rule creation occurs in a workspace where people can compose puzzle pieces representing components for control flow, sensors and actuators, logical operators, variables and advanced features. Despite the high degree of rule customization, the puzzle metaphor makes Zipato promising for non-technical users. A recent systematic literature review identifies the best software tools that allow end users to manage and configure the behaviors of a smart home [19]. Some of the identified tools were also compared on the basis of seven design principles proposed for smart home control.

The analysis of the previous tools highlighted some lacks that make it difficult for non-programmers to use them effectively. In particular, very often the adopted graphical notations for rule specification do not match the mental model of most users [33]. Research on Web mashup composition paradigms – a field that has many aspects in common with smart object composition – showed that graph-based notations are suitable for programmers, while some issues concerning the conceptual understanding of such notations arise with laypeople who do not think about "connecting" services [26, 27, 34].

Another lack is related to the expressive power of the ECA rules that can be specified, which is limited to simple synchronized behaviors. In [7] authors discuss the importance of temporal and spatial conditions to create ECA rules to better satisfy users' needs. Specifying temporal conditions also emerged as an important requirement in home automation to schedule rule for appliance activation [28]. Some tools allow the definition of such conditions only by means of workarounds, for example by considering additional events to monitor the system time, or by creating filters on smart device data (e.g., in Zapier). Obviously, such workarounds complicate the rule creation, thus resulting into a scarce adoption of the available tools, especially by non-technical users, or in their adoption only for very simple tasks.

5 Conclusion

One key aspect in the future of the IoT will be to put in the hands of end user software tools offering natural and expressive paradigms to compose smart objects. Adequate

tools can enable non-expert users to achieve this goal. Task Automation Systems can suit very well the need for synchronizing different objects to program the behavior of smart spaces. The work presented in this paper goes in this direction, as it concentrates on specializing a generic TAS for the composition of services that enable accessing/controlling smart things. The peculiarity of the presented platform is the possibility to adapt easily the composition paradigm. Through a series of user studies we indeed verified that, although in given situations a composition metaphor can result as the most fitting, adaptations might be required in different domains. Sometimes, even the combined provision of different paradigms can result effective.

The composition paradigm currently offered by EFESTO-5W was elicited with the help of end users and then validated by means of controlled experiments. We are therefore very confident that this paradigm encounters the need and capabilities of non-expert programmers, letting them to take advantage of IoT technology. Of course, there are still several aspects to be investigated. First of all, to further extend the capability of EFESTO in supporting the EUD of smart spaces, we are planning future work to understand if and how the addition and the initial configuration of new objects into smart environments could be performed by non-technical users. Actually, our current prototype requires the intervention of expert programmers to define JSON-based object descriptors. We would like to understand whether there can be simple procedures, also based on natural (e.g., gesture-based, proximity-based) interaction paradigms that could (at least partially) enable non-technical users to perform these activities. This implies the identification of a "component model", i.e., a set of conceptual elements abstracting the underlying technology, which can mediate between the technical features to be addressed to program smart objects (the components) and the interaction layer supporting the customization by end users of objects by means of high-level programming constructs.

We also aim to understand how, using recent digital printing technologies, the "fabrication" of smart objects (including the design and production of the physical objects, and the definition of their programming interfaces) can be conducted interactively with the support of visual EUD environments.

References

1. ELASTIC.IO GMBH. Retrieved from http://www.elastic.io/. July 25, 2016
2. Ardito C., Costabile M. F., Desolda G., Lanzilotti R., Matera M., Piccinno A., and Picozzi M. (2014). User-Driven Visual Composition of Service-Based Interactive Spaces. *Journal of Visual Languages & Computing 25(4)*, 278-96.
3. Ardito C., Costabile M. F., Desolda G., Lanzilotti R., Matera M., and Picozzi M. (2014). Visual Composition of Data Sources by End-Users. In *Proc. of the International Conference on Advanced Visual Interfaces (AVI '14)*. Como (Italy), May 28-30. ACM, New York, NY, USA, 257-60.
4. Atooma. Retrieved from https://www.atooma.com/. March 25, 2016
5. Atzori L., Iera A., and Morabito G. (2010). The Internet of Things: A survey. *Computer Networks 54(15)*, 2787-805.
6. AutomateIt. Retrieved from http://automateitapp.com/. March 25, 2016

7. Barricelli B. R. and Valtolina S. (2015). Designing for End-User Development in the Internet of Things. In *International Symposium on End-User Development, IS-EUD 2015*, Díaz P., Pipek V., Ardito C., Jensen C., Aedo I., andBoden A. (Eds.) Lecture Notes in Computer Science, Vol. 9083, Springer International Publishing, Cham, 9-24
8. Bip.io. Retrieved from https://bip.io/. March 25, 2016
9. Cabitza F., Fogli D., Lanzilotti R., and Piccinno A. (2016). Rule-based tools for the configuration of ambient intelligence systems: a comparative user study. *Multimedia Tools and Applications 75(248)*, 1-21.
10. Cappiello C., Matera M., and Picozzi M. (2015). A UI-Centric Approach for the End-User Development of Multidevice Mashups. *ACM Transaction Web 9(3)*, 1-40.
11. Cappiello C., Matera M., Picozzi M., Sprega G., Barbagallo D., and Francalanci C. (2011). DashMash: A Mashup Environment for End User Development. In *Web Engineering - ICWE 2011*, Auer S., Díaz O., andPapadopoulos G. (Eds.) Lecture Notes in Computer Science, Vol. 6757, Springer Berlin Heidelberg, 152-66
12. Casati F. (2011). How End-User Development Will Save Composition Technologies from Their Continuing Failures. In *International Symposium on End-User Development - Is-EUD 2011*, Costabile M.F., Dittrich Y., Fischer G., andPiccinno A. (Eds.) Lecture Notes in Computer Science, Vol. 6654, Springer Berlin Heidelberg, 4-6
13. Coronado M. and Iglesias C. A. (2016). Task Automation Services: Automation for the Masses. *IEEE Internet Computing 20(1)*, 52-8.
14. Costabile M. F., Fogli D., Mussio P., and Piccinno A. (2007). Visual Interactive Systems for End-User Development: A Model-Based Design Methodology. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans 37(6)*, 1029-46.
15. Daniel F. and Matera M. 2014. *Mashups: Concepts, Models and Architectures*. Springer.
16. Desolda G., Ardito C., and Matera M. (2016). EFESTO: A Platform for the End-User Development of Interactive Workspaces for Data Exploration. In *Rapid Mashup Development Tools - ICWE '15*, Daniel F. and Pautasso C. (Eds.) Communications in Computer and Information Science, Vol. 591, Springer International Publishing, 63-81
17. Desolda G., Ardito C., and Matera M. (2017). Empowering end users to customize their smart environments: model, composition paradigms and domain-specific tools. *ACM Transactions on Computer-Human Interaction (TOCHI) 24(2)*, to appear 2017.
18. Desolda G., Ardito C., and Matera M. (2017). Specification of Complex Logical Expressions for Task Automation: an EUD approach. In *End-User Development - Is-EUD 2017*, Markopoulos P., Barbosa S., Paterno F., Stumpf S., andValtolina S. (Eds.) Lecture Notes in Computer Science, Springer Verlag, Berlin Heidelberg (to appear)
19. Fogli D., Lanzilotti R., and Piccinno A. (2016). End-User Development Tools for the Smart Home: A Systematic Literature Review. In *Distributed, Ambient and Pervasive Interactions, in DAPI 2016*, Streitz N. and Markopoulos P. (Eds.) Lecture Notes in Computer Science, Vol. 9749, Springer International Publishing, Cham, 69-79
20. https://ifttt.com/. Retrieved from. Dec 3th, 2015
21. Johnson-Laird P. N. 1983. *Mental models: Towards a cognitive science of language, inference, and consciousness*. Harvard University Press.
22. Li S., Xu L., and Zhao S. (2015). The internet of things: a survey. *Information Systems Frontiers 17(2)*, 243-59.
23. Lieberman H., Paternò F., Klann M., and Wulf V. (2006). End-User Development: An Emerging Paradigm. In *End User Development*, Lieberman H., Paternò F., andWulf V. (Eds.) Human-Computer Interaction Series, Vol. 9, Springer Netherlands, 1-8
24. Lucci G. and Paternò F. (2015). Analysing How Users Prefer to Model Contextual Event-Action Behaviours in Their Smartphones. In *International Symposium on End-User*

Development - IS-EUD 2015, Díaz P., Pipek V., Ardito C., Jensen C., Aedo I., andBoden A. (Eds.) Lecture Notes in Computer Science, Vol. 9083, Springer International Publishing, Cham, 186-91
25. Namoun A., Nestler T., and Angeli A. D. (2010). Service Composition for Non-programmers: Prospects, Problems, and Design Recommendations. In *Proc. of the IEEE European Conference on Web Services (ECOWS '10)*. Lugano (Switzerland), September 14-16. IEEE Computer Society, Washington, DC, USA, 123-30.
26. Namoun A., Nestler T., and De Angeli A. (2010). Conceptual and Usability Issues in the Composable Web of Software Services. In *Current Trends in Web Engineering - ICWE 2010* Daniel F. and Facca F. (Eds.) 6385, Springer Berlin Heidelberg, 396-407
27. Namoun A., Wajid U., and Mehandjiev N. (2010). Service Composition for Everyone: A Study of Risks and Benefits. In *Service-Oriented Computing - ICSOC/ServiceWave 2009 Workshops*, Dan A., Gittler F., andToumani F. (Eds.) 6275, Springer Berlin Heidelberg, 550-9
28. Rode J. A., Toye E. F., and Blackwell A. F. (2004). The fuzzy felt ethnography—understanding the programming patterns of domestic appliances. *Personal Ubiquitous Comput. 8(3-4)*, 161-76.
29. Spacebrew. Retrieved from http://docs.spacebrew.cc/. March 25, 2016
30. Tasker. Retrieved from http://tasker.dinglisch.net/index.html. March 25, 2016
31. http://nodered.org/. Retrieved from. Nov 26th, 2015
32. Tetteroo D., Markopoulos P., Valtolina S., Paternò F., Pipek V., and Burnett M. (2015). End-User Development in the Internet of Things Era. In *Proc. of the Human Factors in Computing Systems (CHI '15)*. Seoul, Republic of Korea, ACM, 2702643, 2405-8.
33. Wajid U., Namoun A., and Mehandjiev N. (2011). Alternative Representations for End User Composition of Service-Based Systems. In *End-User Development - Is-EUD 2011*, Costabile M.F., Dittrich Y., Fischer G., andPiccinno A. (Eds.) Lecture Notes in Computer Science, Vol. 6654, Springer Berlin Heidelberg, 53-66
34. Zang N. and Rosson M. B. (2008). What's in a mashup? And why? Studying the perceptions of web-active end users. In *Proc. of the IEEE Symposium on Visual Languages and Human-Centric Computing (VL-HCC '08)*. Herrsching, Ammersee (Germany), September 15 - 19. IEEE Computer Society, 1550043, 31-8.
35. Zapier. Retrieved from https://zapier.com/. March 25, 2016
36. Zipato. Retrieved from https://www.zipato.com/. March 25, 2016