A Design Space for End User Development in the Time of the Internet of Things

Fabio Paternò and Carmen Santoro

Abstract This paper discusses the issues raised by the Internet of Things for end user development of interactive applications, and how they can be addressed. In such technological setting, applications have to adapt to various types of contextual events, which can be related to users, devices, environments, and social relationships. This calls for environments supporting the development of applications able to cope with dynamic sets of people, objects, devices, and services. The article discusses the characterizing concepts of such environments and their underlying motivations by analysing various solutions proposed to support them and their main design issues. We describe the relevant concepts and discuss how to make them understandable by people without programming experience. One result of this work is a design space, which identifies the main features that should be addressed to support Internet of Things applications using EUD approaches. Such a design space can be used as the basis for comparative discussion amongst various approaches. The analysis provided can also inform the design and development of new tools, and stimulate discussion on current research challenges.

Keywords End user development · Internet of Things · context-dependent applications

1 Introduction

The design and development of flexible software able to match the many possible users' needs is a difficult challenge. One of the main problems is that it is almost impossible to identify all the requirements at design time, since they are often not static (user needs are likely to change over time), and designers also have to

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

C. Santoro
e-mail: carmen.santoro@isti.cnr.it

consider the wide variability of the possible contexts of use. In recent years, the explosion of mobile and Internet of Things technologies has made it possible for people to access their applications from a variety of contexts of use. In this scenario, it is nearly impossible for professional designers and developers to guarantee a good fit between the initially designed system and the actual user needs at any given time. As a result, it is important to design software through methods and tools capable of dynamically and quickly responding to new requirements without spending vast amounts of resources, and able to consider that boundaries between design-time and run-time have become more and more blurred.

End User Development (EUD) is a research field that focuses on enabling people who are not professional developers to design or customize their interactive applications (Lieberman, Paternò, Klann, & Wulf, 2006). Indeed, nowadays users are becoming ever more familiar with ICT technology and they are increasingly capable of using existing tools to create simple applications by themselves. However, since such people usually lack the training of professional software developers, it is simply not possible to use traditional programming environments and methodologies for software development.

The topics related to EUD have already been investigated to some extent in recent years, however, up to a few years ago the main EUD approaches have mainly considered desktop-based applications, such as spreadsheets, unable to adapt to the changing context of use (Paternò, 2013). Only recently have some proposals been put forward to address EUD through mobile technologies. However, the Internet of Things (Atzori, Iera, & Morabito, 2010) introduces further issues such as the need to design how to react to dynamic events that can be generated through a variety of sensors, objects, services, and devices.

If we want to find some relevant aspects in the early end-user development literature before such technology was available, we should look at environments that allowed developers to consider highly interactive applications. In such cases, events were related to user interactions and application functionalities. Alice[1] is a good example of this type of environment. It supports end user development of 3D animations. In particular, it allows users to access lists of events corresponding to user interactions or some specific animation state and indicate what the corresponding event handler should be. HANDS (Pane, Myers, & Miller, 2002) is an environment with similar goals, more oriented to children. It uses the cards metaphor: all objects in HANDS are represented by cards, which have user-defined properties, while the program execution, that is, the manipulation of cards, is represented by an agent. HANDS allows users to select one event from seven predefined event types and indicate the corresponding actions for which it provides some possible operations ("Add," "Sorted," "Sum," "Greatest_Item," etc…). These two important contributions were designed for development in desktop systems and did not consider the variety of events that can be triggered in modern ubiquitous settings.

In the Internet of Things (IoT) vision, "smart" physical objects are networked together, able to interact and communicate with each other, with human beings and/or

[1]http://jupiter.plymouth.edu/~wjt/foundations/alice/Alice05.pdf

with the environment to exchange data and information "sensed" about the environment, reacting autonomously to events in the physical world, and influencing it by running processes that trigger actions and perform services. According to Gartner,[2] there will be nearly 26 billion devices on the Internet of Things by 2020. In this scenario, immense amounts of data can be generated by sensor and communication infrastructures that are growing by orders of magnitude, and IoT applications need to address extremely contextualized user needs. Indeed, one of the primary concerns of IoT is the heterogeneity of devices, sensors, actuators, and services involved in the relevant domains. EUD foresees the use of meaningful logical abstractions and metaphors to abstract out low-level details and make users focus only on relevant aspects, thus facilitating the participation of end-users in the development process.

In order to support EUD of Internet of Things-enabled context-dependent applications we need to consider that context can vary on aspects related to **users** (e.g. tasks, preferences, emotional state), **technology** (e.g. devices, modalities supported, connectivity), **environment** (e.g. light, noise, place), and **social aspects** (e.g. networks, social relationships), and only end users can know the most appropriate ways their applications should react to contextual events. However, to reach such a broad audience of users, authoring environments should be almost transparent to them, presenting a very low threshold to get started, while allowing users to attain high value and even complexity of the software they create or customize. In this way it would be possible to avoid the need to involve in the process people with high developer skills, with the benefit of faster development, better control over the application functionality and improved user experience.

EUD is expected to bring several benefits to the IoT domain. Indeed, giving end-users the adequate tools to create IoT applications is a way to ensure that people's needs will be adequately addressed, and it is also a way to shift innovation from software companies to end users. In addition, EUD foresees the use of intuitive abstractions and metaphors to reduce the cognitive burden associated to handling the multitude and heterogeneity of things, devices, sensors, actuators, services involved in IoT-related domains, and thus support more easily end-user participation in the development process. Furthermore, the use of meaningful abstractions in EUD, in terms of relevant concepts, metaphors, programming styles, vocabularies and intuitive notations should allow different stakeholders (e.g. professional software developers, domain experts, and users) to comprehensively handle the system and also to communicate ideas and concepts. Moreover, EUD is expected to allow users to do more (and more easily) with their existing devices and things within their homes or at work: in other words, EUD for IoT should allow for higher control, more confidence, and better personalisation support. Considering that IoT spams a disparate range of domains, this approach will be key to support the long tail of requirements of IoT end user developers.

In this paper we first discuss metaphors and programming styles for EUD, in particular those more relevant in the IoT field, also providing a discussion of research work that have exploited them. Next we move on to present the proposed

[2]http://www.gartner.com/newsroom/id/2684616

design space, which can be used for comparative analysis. Lastly, we draw some conclusions and mention potential areas that can be object of future research.

2 Metaphors and Programming Styles

We identified two levels for classifying the various techniques that have been mainly used so far in the EUD-related area. One level is represented by **metaphors**, i.e. concepts which do not have any specific connections with the programming world, but rather have a precise meaning in the real world. As such, metaphors' meaning is generally quite familiar to generic users who, by analogy, apply and transfer to the development world the knowledge that the metaphor concept has in the real world. In this way, metaphors provide users with easily understandable cognitive hints expected to facilitate the creation or customisation of an application by decreasing the learning effort needed by a non-professional user to manipulate programming concepts and artefacts. For instance, using the jigsaw metaphor each software component is seen as a piece of a puzzle and the shapes of the various pieces provide the cognitive hints needed to understand the possible compositions.

The second level identified – **programming styles** – is more connected with the programming world and with the need to identify specific interaction paradigms and programming techniques aimed at making end user development easier. At this level we included *programming by example*, *trigger-action –based approaches*, *natural language* techniques, *spreadsheets*, *mashups*, *mock-up –based* and *tangible programming techniques*. It is worth noting that neither level is specific to IoT and, as such, such techniques could be applied to different domains. In the following sections we analyse work in the state of the art which exploited the above concepts, with particular attention to application to the IoT domain (Fig. 1).

2.1 Metaphors

Various metaphors have been considered in this area, Davidyuk, Sanchez, Gilman, and Riekki (2015) discuss the use of some of them. Using the **pipeline metaphor**, applications are represented graphically as directed graphs where nodes correspond to elementary services or activities, and links (i.e. pipelines) connect them. Pipelines can also be organized in complex structures, for example by using logic binary operators connecting nodes, and they can have various implementations depending on how such nodes are represented. Often they are rendered through icons associated with high-level functionalities, with some output and input ports representing the input and the output data, and the application development mainly consists in indicating from where such functionalities receive input and where they send the results of their processing. Realinho, Romão, and Dias, (2012) have proposed IVO (Integrated Virtual Operator), an event-driven workflow/pipeline framework for allowing end users to develop context-aware mobile applications.

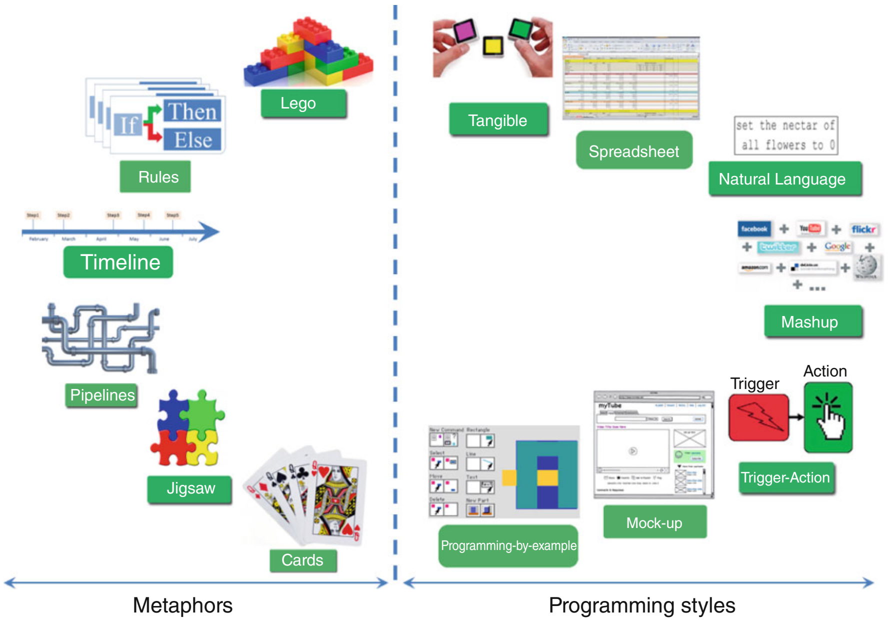

Fig. 1 Metaphors and programming techniques considered in EUD approaches

Using IVO, users build such applications by creating workflows that determine the application behaviour when a specified context is detected. An IVO application is therefore described as a set of workflows that are triggered when some contexts are checked. The workflows are created by combining the available building blocks, which represent the various actions that can be performed by an application. The pipeline metaphor is a visual paradigm which allows for modelling the behaviour of complex applications. However, despite its expressiveness, the use of this metaphor can be problematic when the number of graphical elements and their connections increases, thereby making the resulting diagram difficult to interpret.

In the **jigsaw puzzle metaphor**, each software component is seen as a piece of a puzzle and the shapes of the various pieces provide the cognitive hints needed to understand the possible compositions. Thus, non-expert users can easily associate each puzzle piece with the component it represents. This metaphor has been used in various environments. First, Scratch[3] proposed it for supporting children in learning programming concepts, in particular in creating interactive applications with multimedia content. AppInventor[4] then exploited such metaphor to support the development of functionalities triggered by events in an app user interface. While this metaphor supports more complex configurations than the pipeline

[3]http://scratch.mit.edu

[4]http://appinventor.mit.edu/explore/

metaphor, one disadvantage is that it has limited expressiveness. Indeed, the pieces of the puzzle have a limited number of interfaces (i.e. sides), thereby restricting the set of possible programming expressions. While in Scratch and AppInventor puzzles pieces are used to represent low-level programming constructs, in Puzzle (Danado & Paternò, 2014), they have been exploited to support development of Internet of Things applications on smartphones: the elements are associated with high-level functionalities that can also control actuators. Thus, Puzzle has been designed to facilitate the composition of various pieces through a touch interface for a screen with limited size. Each puzzle piece represents a high-level functionality that can be composed, and its shape and colours indicate the number of inputs and outputs, as well as the information type they can communicate. Thus, the tool provides a usable solution but it is limited to the composition of functionalities for which a puzzle piece has been provided. A similar approach has been investigated also in MicroApp (De Lucia, Francese, Risi, & Tortora, 2012), which exploits graphical composition of common functionalities offered by phone applications, such as taking an image through the camera and saving it, retrieving the contact list, and sending an email. However, in order to work it requires each action or service to expose a description that enables automatically generating the MicroApp puzzle-based user interface. Among commercial systems, Zipato[5] exploits the jigsaw metaphor for rule composition. A variation of the jigsaw metaphor is the *tile-based approach* (Cavallaro, Nitto, Furia, & Pradella, 2010), which allows building programs by combining graphical units (tiles). Although each tile can be combined only with other specific tiles, the shapes of the tiles do not limit the connections. Another variation of the jigsaw metaphor is the *join-the-dots metaphor*, where the editing canvas presents a set of individual devices that are available in the environment. Each device is shown as the centre of a cluster, while the surrounding nodes represent services accessible from the environment. Users create compositions by linking one service to the desired destination device. This metaphor has been applied in the editor of the Platform Composition prototype (Pering, Want, Rosario, Sud, & Lyons, 2009). The main advantages of this metaphor is the simplicity of its visual representation, as only services and devices available for composition are visualized.

Timeline is another relevant metaphor that has been considered in the EUD area. It basically provides a temporal reference along which events/objects are aligned, so helping in organising relevant information in a chronological order. Timelines are typically represented by a line on which various elements are graphically positioned, thus, in timelines temporal relationships (between e.g. events) are basically represented as spatial relationships. TagTrainer (Tetteroo et al., 2015) is an approach exploiting timelines for EUD. TagTrainer enables caregivers to develop rehabilitation exercises for patients with hand or arm mobility problems based on the manipulation of everyday objects. It is designed to create training programs consisting of sequential activities: the workspace located in the center of the screen displays a timeline containing the sequence of all actions associated with the objects involved

[5] www.zipato.com

in the exercise, and, depending on the selected action (e.g. "Place object"), a window on the right side gives the possibility to specify relevant properties, such as the exact location where the object has to be placed at a given time.

Rules represent another used metaphor in the EUD area. The underlying idea is to specify the system behaviour by using a number of if-then statements expressing how the system should behave when specific situations occur. One of the first proposals using rules for EUD was iCAP (Dey, Sohn, Streng, & Kodama, 2006), which introduced the possibility to create if-then rules to support personalization of dynamic access to home appliances. Recently, due to relevancy of contextual dynamic aspects that can potentially affect the behaviour of applications in IoT-based environments, rule-based approaches are receiving increasing interest since end users can easily reason about context and express in rules the desired behaviour of their applications by describing how the application should react to specific events occurring in the context. However, rule-based approaches can become difficult for non-programmer users when complex rules have to be expressed, since e.g. a correct formulation of logical expressions implies the knowledge of some key concepts (e.g. Boolean operators, priority of operators) that could not be always intuitive for non-professional software engineers. Rules can be realised using various programming techniques: in TARE (Ghiani, Manca, Paternò, & Santoro, 2017) rules are expressed using a trigger-action syntax and also by providing a representation in natural language.

2.2 Programming styles

Spreadsheets have proven enormously popular with personal computer users as they provide a concrete, visible representation of data values, as well as powerful features like the possibility to apply formulas to cells, which quickly allow users to solve simple problems within their domain of interest (Burnett, Yang, & Summet, 2002). However, they do not seem suitable to address more dynamic environments such as IoT applications.

The programming style based on user interface **mock-ups** as design tools (Beaudouin-Lafon & Mackay, 2002) has long been considered due to its intuitiveness and effectiveness, and various tools for rapid prototyping for early stages of design, and iterative and evolutionary prototyping have been proposed. They can still be useful in IoT domains as well.

One programming style relevant for EUD is based on **natural language**, a way of programming using a subset of constructs expressed in natural language which should model user's intents. An example approach exploiting this programming style can be found in the work of Perera, Aghaee, and Blackwell (2015), which analysed how a natural language approach can support the definition of policies to manage the home environment. The authors considered the "sticky note" technique for defining the tasks requiring information exchange between IoT appliances and services. The findings revealed mainly that: the average number of words per note was relatively small; people in general adjust their language depending on the type of addressee (human vs. machine); and their technical background affects the way users communicate with machines. Natural language has also been exploited in *composition screens*, where users are able to specify the connections of services and devices for concrete applications, such as in the InterPlay prototype (Messer et al., 2006). InterPlay relies on visual "verb-object-target" constructions which resemble pseudo-English sentences. Users specify a task by first selecting a "verb" (i.e. a command), then an "object" (i.e. content) and, finally, a "target" (i.e. a device). While this approach offers an intuitive user interface, users are only able to trigger the automated composition of the tasks defined in the system at design time. AppsGate (Coutaz & Crowley, 2016) is an EUD prototype which has been deployed in real domestic environments. Its goal is to support end-users defining their own semantics concerning the use of devices and services available at home. AppsGate consists of a server and a set of Web clients. The server is structured in two abstraction levels (one application-agnostic and another one application-specific), and uses OSGi to support the dynamic appearance and disappearance of devices. In order to allow users to express the intended behaviour, a pseudo-natural language is used to express rules that are specified in terms of conditionals (which can regard states and events) and actions. The underlying tool supports a feedforward mechanism to facilitate users in expressing their rules without being burdened by an excessively complicated syntax. In addition, AppsGate also analyses the difficult problem of how to support debugging in EUD environments, by providing the possibility to run programs using a virtual date and time.

Another relevant approach in this area is represented by **tangible interfaces**, where a person interacts with digital information through the physical environment. An example of tangible interactive environment for EUD is in (Truong, Huang, & Abowd, 2004), where they used the fridge magnet metaphor: it mimics refrigerator magnets where the magnets offer a set of words that users can arrange into phrases. It also provides an interface for automated capture and playback (which allows users to replay events that were automatically recorded in the home).

From a HCI perspective, **mashup** refers to a composition of contents and/or features from several sources that determines new client-side interactive applications. For instance, Web mashups can combine data, presentations and functionalities from different Web sites into a novel, single Web application. Various approaches have been put forward in this area. The approach illustrated in (Desolda, Ardito, Matera, & Piccinno, 2015) for mashing up smart things (sensors, actuators) relies on domain-specific customization of the platform. In mashup approaches the basic point is to facilitate new compositions amongst existing components, while a more flexible approach would be to add incrementally new contextual rules for modifying the original behaviour of the interactive application. In (Aghaee & Pautasso, 2014) the authors describe the design and evaluation of NaturalMash, an EUD tool for enabling non-professional users to create mashups by using a subset of natural language expressions, which are associated with mashup components beforehand. They also provided an evaluation in which they compared the expressive power of

NaturalMash with other state-of-the-art mashup environments, showing that their tool offers a good level of expressive power compared with other tools.

IoT is characterized by the presence of a variety of sensors in contexts containing dynamic sets of devices, people, and services. Thus, applications able to exploit such situations need to be informed of the various changes in order to adapt accordingly. This has stimulated renewed interest in **trigger-action programming**, an approach which is mainly based on event-condition-action (ECA) rules. Triggers can be associated with events and/or conditions. Events are instantaneous changes that occur at some point, while conditions define specific contextual states. For example "when the user enters home" is an event since it refers to a state change, while "when the user is at home" indicates a condition corresponding to the state associated with the user being at home. Huang and Cakmak (2015) discuss current Trigger-Action Programming trends and issues. In particular, they found that the distinction between relevant concepts is a source of problems, since users can have difficulties interpreting the difference between events and conditions or between the possible types of actions (for example extended actions, which automatically revert back to the original state after some time, and sustained actions, which do not revert to the original state automatically). Misunderstandings can cause undesired behaviours (e.g. unlocking doors at the wrong time or causing unintended energy waste). Lucci and Paternò (2014) have analysed how three Android apps support this type of programming. Such tools categorize triggers and actions differently according to users' objectives. Their analysis indicates further requirements, for example that EUD tools for IoT should allow the combination of more than one trigger and more than one action in the same rule.

EUD based on trigger-action rules is expected to allow users to do more (and more easily) with their existing devices and things by softening the boundaries between "end users" and "professional developers" as well as between design done before use and software adaptation done at runtime. By specifying customisation rules, users should be able to get better personalisation support and more satisfaction in the use of their context-dependent IoT-based applications. This type of solution can thus contribute to creating technological infrastructures that can successfully establish their usage in practise (Pipek and Wulf, 2009) if they are able to address the specific challenges for obtaining low threshold and high ceiling environments. In (Ghiani et al., 2017) the authors present TARE (Trigger-Action Rule Editor), an environment that allows end users to customize the context-dependent behaviour of their Web applications through the specification of trigger-action rules. The environment is able to support end-user specification of flexible behaviour, including an underlying infrastructure able to detect available devices and objects and possible contextual changes to achieve the desired behaviour. The resulting environment supports the dynamic creation of application versions more suitable for specific contexts of use. An example of its use in a real environment (a students' home) is reported in (Corcella, Manca, and Paternò, 2017).

Another environment that aims to support the development of rule-based reactive applications is IFTTT. It uses the textual syntax "IF This Than That" to specify the scheduling of execution of a certain action (That), and the occurrence of a specified event (This). Its distinguishing feature is that, besides being able to express "recipes" that concern and make changes in the hosting device, IFTTT communicates with widely used Web services, thus allowing the automatic execution of functions related to the internal state of apps such as Facebook, Instagram, Box, Ebay, YouTube and others. In the mobile version the process of creating a recipe is done sequentially through some guided steps. A recent study (Ur, McManus, Ho, & Littman, 2014) found that trigger-action programming can express most desired behaviour in order to customize smart home devices. They evaluated the uniqueness of the 67,169 trigger-action programs shared on IFTTT.com, finding that real users have written a large number of unique trigger-action interactions. Finally, they conducted a 226-participant usability test of trigger-action programming, finding that inexperienced users can quickly learn to create programs containing multiple triggers or actions obtained by extending the IFTTT language, which has limited possibilities, since it only supports applications with only one trigger and one action. This shows that this approach seems suitable to support EUD of context-dependent applications, but needs to be improved in order to allow users to express various desired combinations of events and corresponding actions. Another interesting point of (Ur et al., 2014) is that it shows that the approach based on IFTTT can address emerging Internet of Things (IoT) applications as well. In such applications "smart" physical objects are thought as networked together, able to interact and communicate with each other, with human beings and/or with the environment to exchange data and information "sensed" about the environment, and thereby able to react autonomously to events in the real world, and influence it by running processes that trigger actions and perform services. The availability of mobile tools to perform real time checks of the configuration of on-site visual interactive systems is deemed essential in (Kubitza, Thullner, & Schmidt, 2015) to accelerate the so-called "change and re-try cycles." An example tool for configuring smart environments is described in (Kubitza & Schmidt, 2015). It aims to facilitate physical prototyping by hiding the complexity that arises when many different technologies are combined together. The tool is structured so as to separate the management of devices, events and rules, and mainly targets people with some programming experience since the rules are based on JavaScript.

a CAPpella (Dey, Hamid, Beckmann, Li, & Hsu, 2004) is a desktop tool aiming to address context-dependent applications. It applies the **programming-by-example** style in which the user does not provide the specification of the program but just furnishes examples of sequences of interactions from which the environment understands what the corresponding expected general behaviour is. In this case a user demonstrates context-aware behaviour that includes both a situation and an associated action, and trains the environment on this behaviour over time by giving multiple examples. Once the systems has been trained, the user can run the application, which will then perform the demonstrated action whenever it detects the demonstrated situation. An attempt to apply the programming-by-example paradigm to a mobile development environment is "Keep Doing It" (Maues & Barbosa, 2013), which provides the possibility of identifying context-dependent adaptation rules in the ECA format according to the history of user interactions. The rules are represented through a natural language subset using “when,” “if” and imperatives verbs. An example rule is: “When a wired headset is connected, if my phone is unlocked, launch the Google Play Music application.” A different application of the programming-by-example relevant for IoT is Improv (Chen & Lin, 2017), which aims to support end users in dynamically defining cross-device interactions in order to leverage the capability of additional devices. Thus, users first demonstrates the target UI behaviour using the native input on the primary device. Improv parameterizes the user-demonstrated behaviour. Then, the user demonstrates the input on an accessory device, and Improv associates it with the parameterized behaviour so that the user can obtain the same original application behaviour through the cross-device interaction demonstrated.

3 Design Space

Based on our analysis of metaphors and programming styles, we have identified a logical framework to better understand and compare work in this area. It is composed of seven logical dimensions.

- **Platforms**. The platform supported for the development activities. Traditionally it has been the desktop, but other platforms are being increasingly considered, e.g. mobile, even in combination. e.g. desktop and mobile together (Chen & Lin, 2017);
- **Domains**. An indication of the relevant application domains which the concerned EUD approach can be applied to. The domain can vary depending on the case; in the IoT area examples of application domains often considered are home automation, ambient assisted living, rehabilitation;
- **Events**. In this dimension we consider the types of events that can have an impact on the behaviour of IoT applications. They can concern not only interaction events (i.e. events occurring when interacting with the application), but also contextual events (i.e. those associated to aspects such as user, technology, surrounding environment and social relationships) occurring in the current context of use;
- **Metaphors**. The metaphor dimension aims to analyse the type of representations and interactions adopted in order to make intuitive the specification of the intended application behaviour;
- **Programming styles**. This dimension refer to the programming techniques aimed at making end user development easier for the non-professional user.
- **Actions**. This level describes which type of changes to the application behaviour the EUD environment allows. Different types of actions can be identified, e.g. those performed in appliances (to change the state of actuators), user interface modifications (to change e.g. its presentation, content or navigation), execution of functionalities (e.g. access to an external service like a weather forecast service);

- **Event Compositions/Operators**. This dimension analyses the possibility to build composite expressions of events. Events can be combined in various manners, by using e.g. Boolean operators or temporal operators;
- **Action Compositions/Operators**. This dimension analyses the possibility to build composite expressions of actions. Constructs similar to those occurring in programming languages can be used (e.g. sequence, for, while, if).

Such dimensions can be useful to analyse proposals for EUD environments and think about possible new solutions. Table 1 provides an example of how our logical framework can be used to analyse various proposals. For the sake of brevity we only consider a small set of tools, which have been identified to show different ways to address the design space dimensions.

The first dimension is dedicated to the *platform* supported for the development activities: it can be desktop (as in Alice, HANDS, a CAPpella, TagTrainer) or mobile devices (as in Keep Doing it) or both (as in Puzzle, IFTTT, TARE, AppsGate). As for the application *domains*, some are more oriented to specific sectors (e.g. HANDS for children's animations), while others are more general-purpose (e.g. IFTTT). Regarding the *events*, all the approaches consider interaction events, whereas much fewer approaches consider the full range of event types (interaction, user-related, environment-related, technology-related, social relationships -related), i.e. IFTTT and TARE.

As for the *metaphors*, the most used approaches for addressing IoT domains seem the rule-based one (e.g. IFTTT, TARE, AppsGate) for its immediate way to handle their typical reactive behaviour, and the one based on some subset of natural language (see e.g. Alice and HANDS) for its intuitiveness. The most used *programming styles* were natural language (in Alice, HANDS and AppsGate it was used in an exclusive manner, in TARE it was used in combination with rules), programming by demonstration (a Cappella, Keep Doing it), and the trigger-action approach (IFTTT and TARE).

Regarding the range of *actions* covered by the approaches, it is addressed in a variety of ways and also depends on the considered application domain: TagTrainer is focused on rehabilitation exercises, IFTTT allows users to connect to a predefined set of existing applications, TARE allows the customization of existing Web IoT Applications, AppGate focuses on the home domain, while KeepDoing it aims to extend the possibilities of automating smartphones' tasks. In particular, they cover the modification of the application UI (Alice, Hands exclusively focus on such aspects on desktop platforms), but also consider mobile applications (Keep Doing It, Puzzle), up to covering smart environments and IoT-based settings, especially with the most recent approaches (see e.g. TARE, TagTrainer andAppsGate).

In addition, the possibility of *composing events* in EUD environments has been considered only in a few approaches (a CAPpella, Keep Doing It, TARE and TagTrainer), where in any case a limited set of Boolean operators among AND, OR, NOT have been supported. Instead, *action composition* has been supported in almost all approaches with a few exceptions (namely: IFTTT and AppsGate).

Looking at this table some observations can be derived, also in terms of potential areas that require further research in the near future. For example, while all the

Table 1 Analysis of related work according to the proposed framework

EUD environment	Alice	HANDS	a CAPpella	Keep doing It	Puzzle	IFTTT	TARE	TagTrainer	AppsGate
PLATFORM	Desktop	Desktop	Desktop	Mobile	Mobile/ Desktop	Mobile/ Desktop	Mobile/ Desktop	Desktop	Mobile/ Desktop
DOMAIN	Multimedia animations	Multimedia animations for children	Context-dependent behaviour (e.g. meeting rooms)	Context-dependent smartphone applications	Automate sequences of actions	Composition of various existing Web services	Context-dependent Web IoT applications	Rehabilitation	Home
EVENTS	Interact.	Interact.	Interact.	Interact.	Interact.	Interact.	Interact.	Interact.	Interact.
	Technol.	Technol.	Environ.	User	User	User	User	User	Environ.
				Environ.	Environ.	Environ.	Environ.		Technol.
				Technol.	Technol.	Technol.	Technol.		
						Social	Social		
METAPHOR	Storyboard	Cards	Timeline	Rules	Jigsaw	Rules	Rules	Timeline	Rules
PROGRAMMING STYLE	Natural language	Natural language	Programming by demonstration	Programming by demonstration		Trigger-action	Trigger-action +Natural Language		Natural language
ACTIONS	Handlers associated to interaction events	Handlers associated to interaction events	Smart environment actions	Smartphone tasks	Actions for application UI, objects, appliances, and devices.	Changes in the device, activate Web services and apps	Actions for application UI, objects, appliances, and devices.	Exercises/ actions for physical objects	Behaviour of devices and services available at home
EVENTS COMPOSITION OPERATOR			AND	AND NOT			AND OR	AND	
ACTIONS COMPOSITION OPERATORS	Do together Do together If else While Loop Wait	And Not Or	And	And	Loop		Sequence	Sequence	

approaches consider interaction events, only a few approaches address (at various levels) proper contextual events (e.g. those connected with user, environment, technology, and social aspects). This can be explained with the fact that in the past the initial focus was mainly limited to the events raised by the interactive application, while in more recent years the increasing availability and affordability of various devices and sensing technologies has stimulated the development of context-dependent applications, whose behaviour can be affected by events occurring in the surrounding context. Therefore, the inclusion of various types of contextual events has been mainly considered only in more recent approaches, and thus a more complete coverage of such events in future work would be advisable.

Regarding the actions, we observe a trend similar to the one identified for events: initially the focus was on actions just affecting the interactive application; later on, with the increasing diffusion of IoT technologies and related smart applications, the focus was extended to include actions controlling not only the application but also devices, actuators, physical objects and appliances that can be available in the considered context.

In addition, apart from a few exceptions, the possibility to compose events and corresponding actions is generally very limited, some approaches even do not support their composition at all (as it happens with IFTTT). Therefore, further effort in enabling end users to specify complex expressions of triggers and actions should be pursued because this would provide users with the possibility to specify more flexible behaviours.

However, especially when dealing with complex expressions of triggers (e.g. events and conditions) and actions, there are further aspects that need to be better analysed. As it has been previously highlighted (Huang & Cakmak, 2015), rule-based approaches (and, in particular, trigger-action–based rules) could raise some ambiguity in the interpretation of rules, due to potential inaccuracies in end users' mental models. For instance, interpretation problems can occur when it is not clear whether actions occurring in a rule should be explicitly reverted or not (by using e.g. another rule) as soon as the involved triggers do not hold anymore. This requires further analysis and investigation of the different types of triggers and actions that can be included in complex expressions, in order to avoid such interpretation issues in future EUD tools.

The problem of intuitive composition of logical expressions by end users has also been studied in (Metaxas & Markopoulos, 2017), where an established theory of mental models has been used to guide the design of interfaces for natural programming so that people can find easy to comprehend and manipulate logical expressions. According to such mental model theory people find it easier to conceptualize logical statements as a disjunction of conjunctions (an OR of AND's), as opposed to other logically equivalent forms. Thus, (Metaxas & Markopoulos, 2017) presented a tool which is expected to facilitate end-users in programming context-dependent behaviour using quite complex logical expressions. Although this work represents a useful contribution to facilitate natural programming by decreasing the cognitive load associated with the specification of complex logical expressions, further studies are needed to further elaborate on these key aspects.

Finally, another interesting aspect (yet not fully developed in the EUD area) is how people can test and possibly assess whether the modified/created behaviour of the application actually resulted in the expected one. This need is especially relevant in IoT domains where incorrect behaviour of applications or actuators can eventually have safety-critical consequences (e.g. in the elderly assistance domain and in the home domain). If we consider rule-based approaches, a way to reduce the likelihood of errors in the specification of rules is to allow users to simulate the conditions and the events that can trigger a rule and the effects that they will bring about. An example of this approach can be found in TARE, where users can check the rules (e.g. by simulating them) in order to identify possible errors or conflicts in their specifications, or directly execute them in the current context of use. In this way it should be possible to receive information helpful to find the causes of the undesired behaviour detected and eventually fix them. However, although debugging support could represent an important aid for improving the correctness of the resulting applications, most EUD environments do not include debugging aids for such users (Coutaz & Crowley, 2016) since for non-professional end users debugging becomes especially difficult. Therefore, another possible area that can be subject of possible further investigation is the one dedicated to improve such kind of support in EUD tools.

4 Conclusions

In this paper we have presented a design space, which identifies the main features that should be addressed to support End User Development for Internet of Things applications. The presented conceptual framework is useful to facilitate a better understanding of the important aspects to consider when design EUD environments for IoT, and can be used as the basis for comparative analysis amongst various approaches and inform discussion about areas that can be further investigated.

The discussion about solutions for supporting low threshold/high ceiling specifications of events and actions compositions has still some open points, which require further research work.

Additional aspects that are currently starting to emerge include the possibility for people to test/simulate the behaviour of the IoT applications obtained with the EUD tool in order to assess whether it actually results in the expected one, with the additional possibility to receive information helpful for finding the causes of any undesired behaviour detected and fixing it.

References

Aghaee, S., & Pautasso, C. (2014). End-user development of mashups with natural mash. *Journal of Visual Languages and Computing*, *25*(4), 414–432.

Atzori, L., Iera, A., Morabito, G. (2010). The internet of things: a survey. *Computer Networks*, *54*(15), 2787–2805. doi:10.1016/j.comnet.2010.05.010.

Beaudouin-Lafon, M., & Mackay, W. (2002). Prototyping tools and techniques. In J.A. Jacko & A. Sears (Eds.), *The human computer interaction handbook* (pp. 1006–1031). Hillsdale, NJ: L. Erlbaum Associates Inc.

Burnett, M., Yang, S., Summet, J. (2002). A scalable method for deductive generalization in the spreadsheet paradigm. *ACM Transactions on Computer-Human Interaction*, *9*(4), 253–284.

Cavallaro, L., Nitto, E. D., Furia, C. A., Pradella, M. (2010). A tile-based approach for self-assembling service compositions. In R. Calinescu (Ed.), *Proceedings of the 15th IEEE international conference on engineering of complex computer systems (ICECCS'10)* (pp. 43–52). Oxford: IEEE Computer Society.

Chen, X., & Lin, Y. (2017). *Improv: an input framework for improvising cross-device interaction by demonstration*. New York, NY: ACM TOCHI.

Corcella, L., Manca, M., Paternò, F. (2017). Personalizing a student home behaviour. In *Proceedings IS-EUD 2017, LNCS 10303* (pp. 1–16). Cham: Springer Verlag.

Coutaz, J., & Crowley, J.L. (2016, May–June). A first person experience with end-user development for smart home. *IEEE Pervasive Computing*, *15*(2), 26:39.

Danado, J., & Paternò, F. (2014). Puzzle: a mobile application development environment using a jigsaw metaphor. *Journal of Visual Languages and Computing*, *25*(4), 297–315.

Davidyuk, O., Sanchez, I., Gilman, E., Riekki, J. (2015, December). An overview of interactive application composition approaches. *Open Computer Science*, *5*(1), 2299–1093. doi:10.1515/comp-2015-0007. ISSN (Online).

de A. Maues, R., Barbosa, S.D.J. (2013). Keep Doing What I Just Did: Automating Smartphones by Demonstration. *Proceedings of the 15th international conference on human-computer interaction with mobile devices and services*, MobileHCI 2013 (pp. 295–303). New York, NY: ACM. ISBN: 978-1-4503-2273-7. doi:10.1145/2493190.2493216

De Lucia, A., Francese, R., Risi, M., Tortora, G. (2012). Generating applications directly on the mobile device: an empirical evaluation. In *Proceedings of the International Working Conference on Advanced Visual Interfaces (AVI '12)* (pp. 640–647). New York, NY, USA: ACM. doi:10.1145/2254556.2254674

Desolda, G., Ardito, C., Matera, M., Piccinno, A. (2015, April 19). Mashing-up smart things: a meta-design approach. In *Proceedings of workshop on end user development in the internet of things era – CHI '15 EA* (pp. 33–36). Seoul.

Dey, S. K., Hamid, R., Beckmann, C., Li, H., Hsu, D. (2004). A CAPpella: programming by demonstration of context-aware applications. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '04)* (pp. 33–40). New York, NY, USA: ACM. doi:10.1145/985692.985697

Dey, A.K., Sohn, T., Streng, S., Kodama, J. (2006). iCAP: interactive prototyping of context-aware applications. *Pervasive*, 254–271.

Ghiani, G., Manca, M., Paternò, F., Santoro, C. (2017). Personalization of Context-Dependent Applications Through Trigger-Action Rules. *ACM Transactions on Computer-Human Interaction*, *24*(2), Article 14, 33 pages. DOI: 10.1145/3057861.

Huang, J., & Cakmak, M. (2015). Supporting mental model accuracy in trigger-action programming. *Proceedings of the 2015 ACM international joint conference on pervasive and ubiquitous computing (UbiComp '15)* (pp. 215–225). New York, NY: ACM. doi:10.1145/2750858.2805830.

Kubitza, T., & Schmidt, A. (2015). Towards a toolkit for the rapid creation of smart environments. *IS-EUD*, *9083*, 230–235.

Kubitza, T., Thullner, S., Schmidt, A. (2015). VEII: a toolkit for editing multimedia content of interactive installations on-site. *Proceedings of the 4th ACM International Symposium on Pervasive Displays, 2015* (pp. 249–250). New York, NY, USA: ACM.

Lieberman, H., Paternò, F., Klann, M., Wulf, V. (2006). End-user development: an emerging paradigm. In H. Lieberman, F. Paternò, V. Wulf (Eds.), *End-user development (Human-Computer Interaction Series)* (pp. 1–8). Netherlands: Springer.

Lucci, G., & Paternò, F. (2014). Understanding end-user development of context-dependent applications in smartphones. In *HCSE* (pp. 182–198). Heidelberg: LNCS Springer Verlag.

Messer, A., Kunjithapatham, A., Sheshagiri, M., Song, H., Kumar, P., Nguyen, P., et al. (2006, March). InterPlay: a middleware for seamless device integration and task orchestration in a networked home. In *Proceedings of the 4th annual IEEE conference on pervasive computing and communications (PERCOM'06)* (pp. 296–307). Pisa: IEEE Computer Society.

Metaxas, G., & Markopoulos, P. (2017). Natural contextual reasoning for end users. *ACM Transactions on Computer-Human Interaction*, *24*(2), Article 13. doi:10.1145/3057860.

Pane, J.F., Myers, B.A., Miller, L.B. (2002). Using HCI techniques to design a more usable programming system. *Proceedings of 2002 IEEE Symposia on Human Centric Computing Languages and Environments (HCC 2002)* (pp. 198–206). doi:10.1109/hcc.2002.1046372

Paternò, F. (2013). End user development: survey of an emerging field for empowering people. *ISRN Software Engineering*, *2013*, Article ID 532659, 11 pages.

Perera, C., Aghaee, S., Blackwell, A.F. (2015). Natural notation for the domestic internet of things. In *Proceedings IS-EUD* (pp. 25–41). Cham: Springer Verlag.

Pering, T., Want, R., Rosario, B., Sud, S., Lyons, K. (2009, May). Enabling pervasive collaboration with platform composition. In H. Tokuda et al. (Eds.), *Proceedings of the 7th international conference on pervasive computing (Pervasive'09), LNCS 5538* (pp. 184–201). Nara: Springer.

Pipek, V., & Wulf, V. (2009). Infrastructuring: toward an integrated perspective on the design and use of information technology. *Journal of the Association for Information Systems (JAIS)*, *10*(5), 447–473.

Realinho, V., Romão, T., Dias, A.E. (2012). An event-driven workflow framework to develop context-aware mobile applications. In *Proceedings of the 11th International Conference on Mobile and Ubiquitous Multimedia (MUM '12).* ACM, New York, NY, USA, Article 22, 10 pages. doi:10.1145/2406367.2406395

Tetteroo, D., Vreugdenhil, P., Grisel, I., Michielsen, M., Kuppens, E., Vanmulken, D., et al. (2015). Lessons learnt from deploying an end-user development platform for physical rehabilitation. In *Proceedings of the 33rd annual ACM conference on human factors in computing systems (CHI '15)* (pp. 4133–4142). New York, NY: ACM. doi:10.1145/2702123.2702504.

Truong, K.N., Huang, E.M., Abowd, G.D. (2004). CAMP: a magnetic poetry interface for end-user programming of capture applications for the home. In *Proceedings of Ubicomp* (pp. 143–160). Heidelberg: Springer.

Ur, B., McManus, E., Pak Yong Ho, M., Littman, M. L. (2014). Practical trigger-action programming in the smart home. In *Proceedings of the 32nd annual ACM conference on human factors in computing systems (CHI 14)* (pp. 803–812). New York, NY, USA: ACM. doi:10.1145/2556288.2557420