IoT Cooking Workflows for End-Users: A Comparison Between Behaviour Trees and the DX-MAN Model

Filippos Ventirozos*, Riza Batista-Navarro†, Sarah Clinch‡ and Damian Arellanes§
*†‡Department of Computer Science, University of Manchester, UK
§School of Computing and Communications, Lancaster University, UK
Email: {filippos.ventirozos, riza.batista, sarah.clinch}@manchester.ac.uk, damian.arellanes@lancaster.ac.uk

***Abstract*—A kitchen underpinned by the Internet of Things (IoT) requires the management of complex procedural processes. This is due to the fact that when supporting an end-user in the preparation of even only one dish, various devices may need to coordinate with each other. Additionally, it is challenging—yet desirable—to enable an end-user to program their kitchen devices according to their preferred behaviour and to allow them to visualise and track their cooking workflows. In this paper, we compared two semantic representations, namely, Behaviour Trees and the DX-MAN model. We analysed these representations based on their suitability for a range of end-users (i.e., novice to experienced). The methodology required the analysis of smart kitchen user requirements, from which we inferred that the main architectural requirements for IoT cooking workflows are variability and compositionality. Guided by the user requirements, we examined various scenarios and analysed workflow complexity and feasibility for each representation. On the one hand, we found that execution complexity tends to be higher on Behaviour Trees. However, due to their fallback node, they provide more transparency on how to recover from unprecedented circumstances. On the other hand, parameter complexity tends to be somewhat higher for the DX-MAN model. Nevertheless, the DX-MAN model can be favourable due to its compositionality aspect and the ease of visualisation it can offer.**

***Index Terms*—Internet of Things, Behavior Trees, DX-MAN model, End-User Development**

I. Introduction

Numerous IoT End-User Development paradigms and applications have emerged in recent years [1]. Most of these applications address straightforward domestic tasks such as switching a light on when a door opens [2]–[4]. However, they have not addressed how to support end-users in programming IoT cooking workflows.

We make the case that IoT cooking workflows differ from other IoT or robotic workflows in three ways: (1) there is a plethora of possible workflows (i.e., cooking recipes), many of which are subject to multiple constraints such as the user's taste preferences, diet restrictions, time availability and mood, available devices and ingredients; (2) many of these constraints may be fuzzy; and (3) IoT cooking workflows are subject to complex dependencies between heterogeneous kitchen devices that need to work in synergy with the end-user, making it difficult to combine them. Based on the above, we argue in this paper that the main architectural requirements that need to be addressed are variability and compositionality.

Our research question is "What is the most suitable semantic representation for handling IoT cooking workflows in the context of end-user development?". We examine two semantic representations that have been proposed for representing IoT workflows. First are Behaviour Trees (BTs) which originate from the domain of gaming, and have made their way to robotics and have been proposed for IoT and cooking [5]. BTs are a type of abstract state machines. They have been successful in the gaming world since they allow for programming NPC[1] behaviours with an intuitive design that is easy for designers to track, omitting the drawbacks of the previously used finite-state machines (i.e., the lack of scalability, maintainability and re-usability) [7], [8]. Second is the DX-MAN model. DX-MAN was specifically created to handle the behaviour of multiple and heterogeneous IoT devices. It accomplishes that by using hierarchical composition operators for composing two or more services into more complex ones [9]–[11]. We acknowledge that other semantic representations have been used for ambient intelligent environments such as Petri Nets [12] and their derivations [13], Business Process Model Notation (BPMN) [14], [15] or task models [16] but we leave those as part of our future work. Our investigation of BTs and DX-MAN is driven by the aim to identify a machine-readable representation for end-user programming/development (EUD)[2] in the context of an IoT kitchen. The most important human factor that we considered as we carried out our analysis is end-user expertise. Within the context of an IoT kitchen, this corresponds to an end-user's cooking skills and preferences.

We reviewed smart-cooking user requirements (identified in previous studies) to investigate how workflows produced by less expressive EUD frameworks (e.g., Event Condition Action and Block-based programming languages—introduced in the next section) can be parsed into each of the two more

This research is supported by ARM Ltd and the UK EPSRC under grant number EP/S513842/1 (studentship 2109081).

[1]Non-Player Character, defined as any character in a game which is not controlled by a player [6].

[2]We use the terms end-user programming and end-user development interchangeably since we refer to various levels and expertise of end-users in the development cycle [17]. For brevity, we will refer to them as EUD.

expressive and more machine-readable semantic representations, i.e., BTs and DX-MAN. We then compare them using cognitive workflow complexity estimates. Also, we compare visualisations of the two semantic representations.

The remainder of this paper is structured as follows: in Section II, we analyse user requirements of (smart) kitchen end-users. In addition, we introduce two popular EUD metaphors and the notion of different skill levels of end-users and various stakeholders in cooking workflow development (i.e. meta-design). In Section III, the two semantic representations (i.e., BTs and DX-MAN model) are introduced. In Section IV, we compare the semantic representations with each other, analysing how each can be visualised and how they can accommodate EUD metaphors. We finally evaluated them for workflow complexity [18].

II. Background

In this section, we present some background on end-user requirements for smart kitchens. We also introduce the meta-design EUD framework. Finally, we describe two EUD metaphors and their common implementations.

A. Smart Kitchens and End-User requirements

A smart kitchen environment, in contrast with the rest of the domotic environments, can be highly diverse. Consider a lighting control device that determines the need for light by checking for motion or occupancy. It works well when user behaviour aligns well with a set of actions and triggers. For pre-configured sets of IoT devices, this entails that the behaviours are sufficiently similar to fit the set of triggers appropriately. However, kitchen use is highly diverse [19]. Cooking by itself often requires complex chains of actions (e.g., mixing at different speeds at different times, heating in stages), making it challenging to ensure that the technology can anticipate and meet the needs of its users.

Smart kitchens need to know the workflow that is being executed, the entities (i.e., ingredients and devices) involved and their states in each cooking step. Specifically, it has been argued that when performing a cooking workflow[3], users need specialised technology to: 1) remind them of where they were in the workflow in case they were distracted 2) weigh ingredients accurately 3) know what ingredients they currently have 4) locate items within their environment (especially needed in small kitchens) 5) utilise labour-saving devices 6) access a virtual assistant that provides advice or guidance [19], [20] From these requirements, it is apparent that kitchen IoT devices need state tracking to "know" in every step what ingredients have been processed, how they have been processed and how to assist the user in performing future steps with the available ingredients and devices.

The purposes of different devices within a smart kitchen can be heavily intertwined. As stated by [20], users need different kitchen devices (e.g., oven, stove, fridge and cooking hood) to be able to cooperate with each other (e.g., start the cooking hood when the stove is in use), unlike the rest of domotic devices which are not related to cooking.

In the following section, we consider three architectural requirements for the representation of IoT cooking workflows.

B. An abstract IoT Cooking Workflow should respect:

1) Workflow Variability: An IoT cooking workflow needs to accommodate the availability of devices and all the possible behaviours of its users. An IoT workflow should be able to recover and implement an alternative workflow if certain devices do not respond. If the user changes their mind about the recipe at any point, innovating a new workflow should be considered. The ability to create system variants for different users or contexts of use is called **variability** [10], [21].

A framework with high variability should be capable of accommodating online changes. Should the IoT workflow constructor not be able to accomplish a goal (or near-to-goal state) due to user absence, for instance, the framework should be able to bring the workflow to a safe state avoiding as much as possible: food expiry, food contamination or any type of hazard. Specifically, we can expect the following user "degrees of freedom" which can happen sequentially with each other: • start a new recipe on its own or parallel with another recipe being executed • cancel a recipe at any point • start cooking without a selected recipe • add an additional step to a current recipe or skip a step in a recipe • change the order of the cooking recipe steps • tweak a step, deviating from the recipe • add new constraints (e.g., number of guests, a new device of preference). Moreover, with the inclusion of more labour-saving devices (as per the user requirements outlined in Section II-A), the number of combinations of possible IoT cooking workflows is increased. Fully or partially automating cooking steps with these devices and can easily lead to a combinatorial explosion problem [22].

In addition, recent methods in machine learning on procedural data can aid in constructing exhaustive IoT cooking workflows to fulfil a user's requirements. It has been hypothesised that AI technologies can teach smart environments how to behave by setting rules automatically. In the vision paper of [23], it was recognised that the vast amount of online resources such as videos can be used to drive ubiquitous computing. In fact, the study of [24] suggests harnessing videos to segment its frames (cooking steps) and transform them into a latent space where a Markov Decision Process algorithm is employed to learn the sequence of cooking steps—essentially a planning algorithm that can potentially narrate a cooking workflow.

Over time, it has become more apparent how machine learning can aid in creating IoT cooking workflows and setting rules for the user automatically. However, the downside is that visualising a plethora of parsed workflows is an overwhelming task for any end-user. By demonstrating semantic support to encapsulate multiple computational behaviours in a single entity, workflow variability plays an important role in taming the combinatorial explosion of IoT cooking workflows.

2) Fuzzy Parameters: An ideal IoT workflow representation should be able to accommodate fuzzy parameters. According

[3]We refer to it as workflow since it involves a series of steps which may or may not be dependent on each other.

to [25], IoT for smart home environments need to be affective. They need to act depending on contextual variables that may not be binary. In addition, cooking is probably the most fuzzy environment due to parameters such as taste preferences and quantities (e.g., "a bit of salt").

3) Compositionality: A recipe involves a series of actions. These involve either human handling or device automation. We denote the range of actions in an IoT cooking workflow as nodes in a graph. Each node documents which ingredients have been processed and what is allowed in terms of changing or tweaking that cooking action. Consequently, several components and parameters can become applicable, including: when to start the action (e.g., when the previous IoT device sends a signal, when a certain amount of time has elapsed or when a user issues a trigger), and the potential device settings (e.g., temperature, time, mode). Moreover, the devices used might include those which deviate from standard ones found in the kitchen. Examples include: a robotic arm[4], oven cameras [26], tracking-recording cameras [27]–[29], automated ingredient scales [27], autonomous stoves [27], [28], accelerometers and RFID readers attached to kitchen objects [29]. The investigation of how these workflows are automatically formed pertains to the field of automated planning [30], which is out of the scope of this paper.

Inside each node, multiple functions can take place. Various checks such as "Did the user already perform this action?", "Should it be adjusted to user preferences?", "Does the user want to perform this cooking step by themselves?", "What kind of guidance does the user prefer?", "Is the user still in the kitchen or are they away?". Consequently, for each cooking step involving a device, there are numerous checks that might be necessary. Thus we hypothesise that a suitable IoT workflow representation should be capable of encapsulating each sub-component of each device-node according to the level of expertise of each end-user (novice to advanced).

An end-user may also desire to add new functionalities to a cooking workflow. For instance, as mentioned in the user requirements in Section II-A, a user may want to add a new service to find their tools. Another example would be a user adding a new labour-saving or precision smart device, e.g., a smart cooking hood that switches on automatically when the hob is used or when the user specifies so (as it might be the case that for a short cooking duration a user prefers to not be disturbed by the hood's noise).

A desirable cooking workflow can accommodate any modular change in its flow without affecting other workflow parts. The ability of a system to compose different services into more complex ones without resulting in unexpected behaviours is referred to as **compositionality** [9].

C. End-User Development for IoT

1) Meta-Design: A meta-design EUD framework refers to the continuous development of personally meaningful socio-technical systems with multiple stakeholders (including end-users) who are actively engaged in the process [31]. Characteristically, it could refer to designers, computer scientists, and engineers (who are not experts in the problem domain) and end-users who are experts in the problem domain, but unaware of the software solutions [32]. Similarly, smart-cooking can involve a range of multiple stakeholders and end-users, from software engineers, device vendors, system specialists to various types of end-users including domain specialists (i.e., chefs) and people who are still learning how to cook. To some extent, cooking is an art, and as such, the users involved in it may not only replicate recipes that are already known (i.e., read-only culture) but may also introduce alternatives or enhancements (i.e., read-write culture) [33] as well as customise recipes to fit daily practical personal needs (i.e., allergies, preferences). The key element for such an ecosystem to work is the multiple levels of end-user programming/development. In this paper, we investigate a three-tier EUD framework: (1) Event Condition Action or ECA; (2) Block-Based visual programming language; and (3) the semantic workflow representation which forms the basis of the IoT cooking workflow; these representations are discussed in detail in Section III.

2) EUD Models: Below we describe the most commonly used EUD metaphors and their implementations. These are analysed in Section IV.

a) Event Condition Action: The most widely known programming paradigm is based on the Event Condition Action framework [25]. One of the most popular applications which follow this is IFTTT[5] which stands for "IF This happens Then do That", i.e., an action is executed if a certain condition is met. It has been used in several studies focussing on smart homes [3], [34], [35]. Other smart home approaches build upon enhancements to the IFTTT model [2], [36], [37]. Although the Event Condition Action paradigm is appealing due to its simplicity, research suggests that establishing rules from end-users can lead to errors. Specifically, new rules can conflict with older rules leading to unpredictable and dangerous behaviours (e.g., front door that is unexpectedly unlocked) [38], [39]. In addition, this paradigm does not offer more verbose editing options like the construction of loops and more sophisticated workflows.

b) Block-Based Visual Programming Language: Visual programming languages have the potential to enable the user to code more complex behaviours. According to a recent systematic review, the most used visual programming language overall and in the context of IoT programming is block-based programming [40]. Block-based visual programming tools allow users to construct a program by combining together visual blocks together like a jigsaw puzzle. These blocks can represent code idioms such as variable setting and loops [41], [42]. One should consider that block-based programming languages are by no means the ultimate desiderata for end-user development. [43] argue that in programming humanoid robots, block-based solutions seem to work well when the

[4]https://misorobotics.com/

[5]https://ifttt.com/

possible options in a scenario are quite limited. However, this is not true with modern humanoid robots which can flexibly react to many possible events and perform a wide variety of actions. We highly suspect this statement holds true in IoT cooking workflows as well, since an overwhelming set of parameters and tasks can be involved, as described earlier in Section II-B1: Workflow Variability.

III. Methodology

We analyse two semantic representation models: Behaviour Trees and the DX-MAN model. First, we introduce each of them and how they can be adapted for IoT cooking workflows. We then examine each in terms of: (1) how they can be flexibly visualised by the end-user, and (2) how a program developed by an end-user using Event Condition Action or Block-Based Visual Programming Languages, can be parsed into a comprehensible representation. The user requirements we identified in the preceding section allowed us to formulate hypothetical scenarios which then form the basis of our comparison of these two models in Section IV.

A. Behaviour Trees

A Behaviour tree (BT) is a directed acyclic graph consisting of different kinds of nodes. It is a tree-shaped graph, hence the name behaviour tree. A BT consists of a set of nodes. At execution, the top node is executed and in turn, it will try to execute its child nodes. When executed a node will return one of a number of status codes, namely, `success`, `failure`, and `running`. While the first two are self-explanatory, `running` denotes that the node, or one of its child nodes, is currently under execution. A BT consists of different types of nodes, i.e., leaf and internal nodes [8], [44].

a) Leaf Nodes:

- Action - An action represents a behaviour that the device can perform. When an action is completed, the node returns `success` or `failure` depending on whether the execution was successful. Otherwise, the node returns `running`. Actions are depicted as rectangles or rounded rectangles.
- Condition - A condition checks the state of the environment, i.e., a proposition. It returns either of `success` or `failure` depending on whether the proposition holds. The evaluation of a condition is instantaneous and hence will never return with a `running` status. A condition is depicted with an oval shape.

b) Internal Nodes:

- Sequence Node - A sequence node tries to perform all its child nodes in sequence from left to right. If one of the child nodes fails, the sequence node will return with a `failure` status. Otherwise, if every child node has been successfully executed, it will return `success`. A sequence node is depicted as a square with an arrow inside it, i.e., $\boxed{\rightarrow}$.
- Fallback Node - A fallback node executes its children nodes from left to right. If a child node returns `success`, the fallback node also returns `success`. If a child node fails, then the fallback node will try to execute the child node that is next according to the left-to-right ordering. It returns `failure` if and only if all its children return `failure`. A fallback node is depicted as a square containing a question mark, i.e., $\boxed{?}$.
- Parallel Node - A parallel node executes all of its children and it returns `success` if M children return with the same. It returns `failure` if $N - M + 1$ children return `failure`, and returns `running` otherwise. N is the number of children and M is a user-defined threshold. A parallel node is depicted with two arrows, one on top of the other, inside a square, i.e., $\boxed{\rightrightarrows}$.
- Decorator - A decorator node is a type of unary controller that imposes some restrictions on the executions of its single child node. For instance, one decorator may be defined to ensure that an action is not repeated more often than every fifth second. If the node is called sooner than five seconds after it has last been executed, it returns `failure`. Decorators are depicted as a rhombus with descriptive text (or δ for short), i.e., $\diamond\!\!\boxed{\delta}\!\!\diamond$.

The above is regarded as the vanilla version of BTs. In the literature, various methods have augmented the number of internal nodes. This was done by including, for instance, a repeat internal node for non-deterministically selecting leaf nodes, an `always-failure` node that will return success when only a leaf node is in a `running` state [45], or a repeat node (which selects leaf nodes until a number of successful runs have been reported) and a reset node which resets a child node up to a number of times upon failure [46]. These are out of scope in our study.

To adapt BTs for the kitchen IoT workflow application, we use the following straightforward architecture. Firstly, it is expected that some tasks are static and others are dynamic (e.g., setting the device parameters according to the ingredients used). Secondly, we hypothesise that to deploy such an application, certain aspects are common between different devices, e.g., when to trigger a device, what to do if a cooking step has been completed, should a step be automated or not according to the preference of the user). These "sub-BT" components differ per device type. For instance, a cooking task might involve sensors (e.g., a smart scale or an inspection camera) which can be triggered by themselves and know when a certain cooking step has been initiated due to their sensing capabilities. However, actuators would need to be triggered by another entity.

Given the above, we set each device (device, sensor or actuator) to be on standby until it is invoked by a Complex Event Processing (CEP) unit to process a recipe or is triggered at any point by the user. Then, various probabilities can be assigned for the invocation of each device. These can be assigned from a machine learning module based on the user's selected recipe and/or their previous preferences. In Figure 1 one can visualise a BT with a CEP with probabilities assigned per device task.

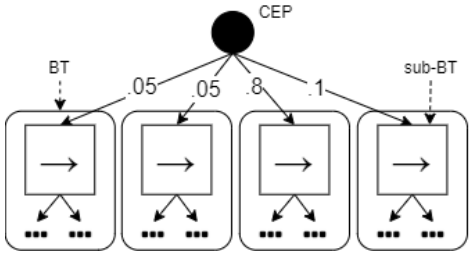


Fig. 1: The CEP can trigger each BT according to the probability estimated by an estimator, and waits to hear confirmation of either `success` or `failure`. Each BT symbolises a different cooking action involving device(s) or user guidance, and can contain sub-BTs.

B. DX-MAN Model

DX-MAN [9]–[11] is an algebraic composition model where services and composition operators are first-class semantic entities. On the one hand, services are spaces of workflow variants. On the other hand, composition operators hierarchically compose two or more services into more complex ones. This process is known as algebraic composition, and it is rooted in single-sorted algebra in which mathematical objects of one type are hierarchically composed into new objects of the same type [22].

a) Services:

- Atomic Service - An atomic service has an invocation connector and a non-empty finite set of primitive operations (which are akin to BT actions). An invocation connector is similar to a BT internal node and it is responsible for selecting the appropriate primitive operation to be invoked.
- Composite Service - A composite service has a composition operator CC to compose services which can be either atomic, composite or any combination thereof. Semantically, it is equivalent to a workflow space W which is a (potentially infinite) set of workflow variants. These variants represent alternative control flows for the execution of the composed services, and there are operators for defining sequential, branchial and parallel workflows.

b) Operators:

- Sequencer Operator - A sequencer operator SEQ uses the Kleene star operation to allow the repetition of n elements (i.e. operations and/or other workflows), resulting in infinite sequences. It then defines a workflow space W_{seq} for a composite service. Each $w_{i\in[1,\infty]} \in W_{seq}$ is a sequential workflow.
- Paralleliser Operator - A paralleliser operator PAR allows the execution of multiple elements in parallel. As it supports element repetition, it defines a workflow space W_{par} for a composite service. Each $w_{i\in[1,\infty]} \in W_{par}$ is a workflow executing multiple elements in parallel.
- Exclusive Selector Operator - An exclusive selector $XSEL$ defines a workflow space W_{xsel} with $2^m - 1$ exclusive branchial workflows for a composite service. Each workflow $w_{i\in[1,2^m-1]} \in W$ contains at least one element out of m possibilities, and chooses a single element to be executed.

In [11], DX-MAN was extended with the notion of feedback control loops for composite services. Particularly, Monitoring, Analysis, Planning, Execution and Knowledge (MAPE-K) [47] allows the dynamic selection of workflow variants depending on the given context. The MAPE-K would act similarly to the hypothesised CEP of the BTs. In Figure 2 one can see the MAPE-K components and a possible abstract IoT cooking workflow.

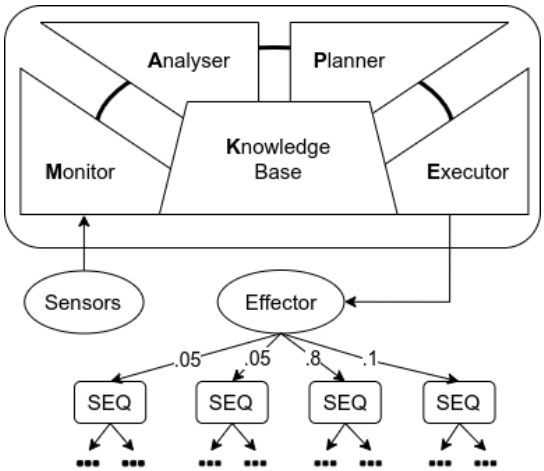


Fig. 2: The MAPE-K model monitors the sensory input, analyses then plans the abstract workflow structures, shown below with the operators (i.e. SEQ). Next, the executor and the effector determine the variables for each operator and can assign probabilities to each Sequencer operator to execute, similarly to Figure 1.

IV. Evaluation & Discussion

We evaluate Behaviour Trees and the DX-MAN model in terms of two aspects that concern the end-user: the ability to allow the end-user to visualise IoT cooking workflows, and the extent to which the architecture facilitates EUD. In relation to these two aspects, we utilised use cases driven by user requirements (Section II-A). It is also worth noting that these cases predominantly require the architectural requirements of variability and compositionality. Then, to compare between these semantic representations, we used the three measures of workflow complexity introduced by [18]. These measures of complexity were designed to express a quantitative perspective on how a human administrator (end-user) perceives workflows. These three measures were inspired by the metrics traditionally used to evaluate software complexity based on control flow, data flow and space complexity [48], which are respectively, execution complexity, parameter complexity and memory complexity. Below we provide an overview of each of them.

- Execution Complexity: This typically refers to the number of actions. In BTs we consider the number of leaf nodes and in DX-MAN, the number of operations. This metric also refers to the number of context switches. Specifically, whenever an action is in the same configuration context as any other action (noting that consecutive actions belong to the same execution workflow), then there is no increment of the context switch parameter. However, if it the action is in a different context, then the value is increased to the total number of configurable contexts (e.g., if an action changes its behaviour or is not executed because of a workflow element).
- Parameter Complexity: The number of parameters used. Usually, they are expressed at the level of internal nodes/operators.
- Memory Complexity: The number of parameters defined by end-users but not used for immediate action. Instead, the end-users need to remember them. The complexity increases if more intervening parameters need to be memorised over time and if these parameters change due to contextual variables.

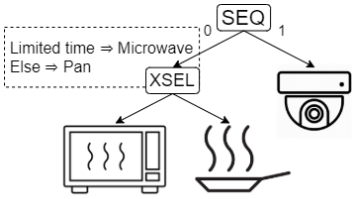
A. End User Visualisation

Visualisation is a primary concern for the end-users since they need to observe and be aware of how their IoT cooking workflow has changed according to various parameters. Below, we discuss the extent to which BTs and DX-MAN support the visualisation of architectural properties of IoT workflows.

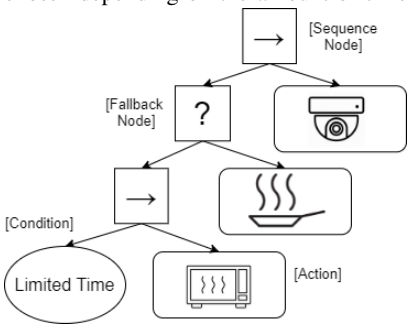
1) Workflow Variability: A challenging aspect of IoT cooking workflows is the plethora of parameters that can be applied, which results in multiple workflow variations. In Section II-B1, we detailed some reasons for accommodating multiple workflows.

The DX-MAN model provides the exclusive selector module (described in Section III-B) which can hold multiple conditions to choose the remaining path of the workflow, in any combination. Let us consider a simple workflow whereby a recipe requires: using the microwave if time is limited; otherwise, a pan; and lastly, a zenith inspection camera is used for monitoring. The DX-MAN model uses the sequence and the exclusive selector operator with its parameters to execute the workflow, as shown in Figure 3a with some possible visual notations. In BTs, the different options for parameters are structured with condition states and further forks. For instance, the same example is demonstrated as a BT in Figure 3b.

From the above example, one can derive the following workflow complexities. In the DX-MAN model, there are three operations whereas there are four leaf nodes in the BT. Furthermore, the DX-MAN workflow has one context switch, i.e., the workflow changes depending on the time variable. In the BT example, there are two context switches: the microwave action may be triggered or not depending on the time variable, and also, the use of the pan may be triggered or not depending on the same statement. Hence, the end-user needs to comprehend two context switches instead of one. Nonetheless, the BT does not carry any parameters, whereas,



(a) A DX-MAN workflow example. It uses a SEQ operator which prioritises the left child (XSEL). After the XSEL operator, the suitable equipment is chosen depending on the amount of time available.



(b) A BT example where the sequence node on top dictates the actions and conditions to take place from left to right. On the left is a fallback node which has a sequence. Underneath it is the condition, i.e., if time is limited, then the microwave should be used. If that fails, the fallback node comes in, i.e., the pan should be used. After that an inspection camera is used.

Fig. 3: Possible implementation of an example workflow based on DX-MAN (Figure 3a) and a BT (Figure 3b).

the DX-MAN model has one parameter under the XSEL (for the time variable).

Lastly, in this example, there is no memory complexity in either of the representations. In conclusion, we believe that the DX-MAN model is more favourable since the parameter complexity can be easily reduced by obscuring the parameters that the end-user is not interested in, whereas the same cannot be said for execution complexity. In addition, generally, the XSEL module provides more information than a BT internal node, since it can select different actions and orders based on the constraints, whereas the BTs need to demonstrate verbosely every possible path resulting in more execution complexity.

In the literature, there have been implementations for visualising multiple workflows. A popular method is the node-link graph [49]. The AVOCADO visualisation tool [50] follows this principle [49]. It compresses nodes from data provenance workflows to make it easy for a user to track them (see example in Figure 4). The user can also select which workflows to visualise. Similarly, cooking workflows can be queried by the end-user and visualised accordingly.

A BT implementation can also be visualised using the AVOCADO technique but with some challenges. The BT works with sequential selections, every time asserting and executing from left to right. Consider Figure 5 which depicts another example workflow: making dough can sometimes require

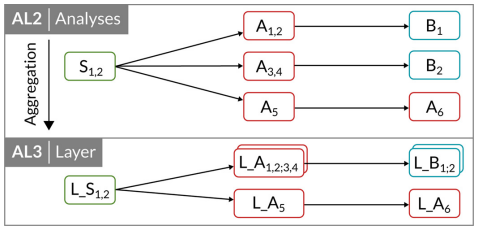


Fig. 4: A workflow compressed using AVOCADO (aggregation level into layers using motif-based aggregation) [50]. The visualisation can be implemented for the DX-MAN model, where the $S_{1,2}$ would represent an Exclusive selector operator (XSEL) followed by Sequencers (SEQ) with some A_n, $B_{n'}$ operations.

whisking multiple times then adding water, and then whisking again. With its fallback node, the BT provides an elegant way to ensure that the dough is never too dry. Nonetheless, it will be challenging to represent such a direction-agnostic sequence in AVOCADO.

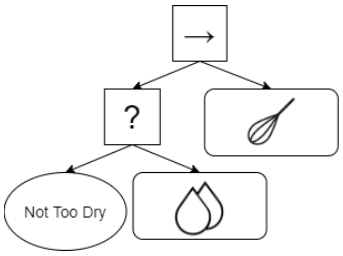


Fig. 5: A BT example where the sequence can go backwards. While whisking to make the dough, in every time step it has to hold true that the dough is not too dry; if it is then water needs to be added before whisking again. This can lead to a non-sequential series of actions: whisk, add water, whisk.

Meanwhile, due to its workflow direction being deterministic, the DX-MAN model does not suffer from the aforementioned issue in BTs.

2) Fuzzy Logic Visualisation: As discussed in Section II-B2, IoT cooking workflows may have variables that are fuzzy. With its exclusive selector operator, the DX-MAN model can support multiple actions depending on the fuzzy constraints (as in Figure 3a, there may be multiple options under the XSEL which pertain to the time variable). In contrast, BTs need to model this with nested BTs (as in Figure 3b where each new option would require a combination of fallback, sequence nodes and conditions). Thus the DX-MAN model is more favourable in terms of execution complexity since there can be a minimal set of exclusive selectors handling the fuzzy variables. Nonetheless, if the application would require multiple conditions on the exclusive selectors then it would increase its parameter complexity.

B. End User Development

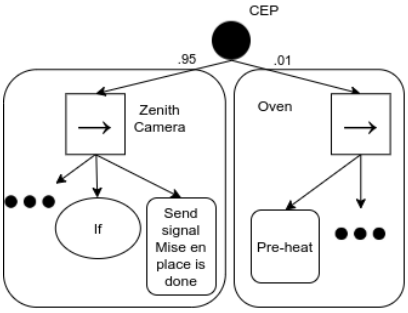
In Section II-C, we mentioned the two most popular EUD frameworks which correspond to a slightly different level of end-users. Firstly, we discuss the ECA framework, and the possible use cases that it can support. These cases explore the variability architectural requirement. Secondly, we mention how a block-based visual programming language can be utilised for creating IoT cooking workflows. Also, in that framework the cases of variability and compositionality are evident. In each scenario we compared BTs and DX-MAN.

1) Event Condition Action:

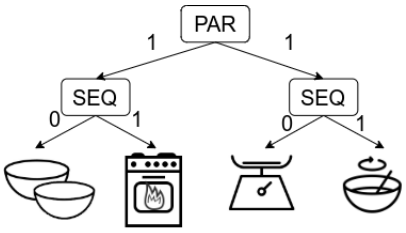
a) Adding Device Interdependencies: Let us consider an example case whereby a user wants to start pre-heating the oven when they have prepared their ingredients (*mise en place*), a case of intertwined devices. To perform this action with the BT architecture two possible edits need to be made. One is to update the zenith inspection camera to detect when the *mise en place* has finished and to publish a signal to the CEP that it has finished, resulting in new leaf nodes. In the oven trigger sub-BT, a leaf needs to be added to receive the completion signal from the CEP. Figure 6a depicts a possible architecture; all the other actions (e.g., heating the oven) are omitted for simplicity. It is noticeable that adding many of these event-condition-action rules makes it challenging to track them in the BT architecture and requires making edits inside each sub-BT.

Meanwhile, via an exclusive selector and a paralleliser operator, the DX-MAN model creates the workflow shown in Figure 6b. One might argue that it should be possible to implement a similar structure in a BT by adding a Parallel Node and a Sequence inside each to create a similar structure to the DX-MAN model. This method would require that all of the services have a flat hierarchy, which is unlikely since a developer would most likely need to group some sub-BTs and create dependencies. The only exception is when every time the whole BT is created from a flat hierarchy of services, in which case the DX-MAN model can be replicated. Nevertheless, the DX-MAN model will have an edge over composing services with re-using workflows via its algebraic composition mechanism. Specifically, the DX-MAN model benefits from this mechanism in the sense that all of the services are flat in the hierarchy (atomic services, i.e., services offered by each device that can be used in combination with other devices, as demonstrated also in this example). Then, the operators configure the workflows dynamically and re-use some already established ones via the MAPE-K loop. Thus, if an end-user would use an ECA framework to create dependencies between devices, we hypothesise that a developer/domain expert would prefer the DX-MAN model since it would have have fewer number actions, context switches, and hence, less execution complexity.

b) Replacing Device: Another task that a user may be required to perform is to add a new device to already existing cooking workflows or assign recipe actions to a new labour-saving device, as described in Section II-A. Let us consider the example that a user has purchased an airfryer and wishes to update their IoT cooking workflows to replace the oven with their airfryer. An ECA would be required to express this preference. This can be: if $oven < functionality =$



(a) Following the dynamic BT allocation architecture described in Section III, the CEP selects which sub-BTs will be executed. Inside each sub-BT there are new conditions that need to be assessed to send a signal to the CEP to then trigger the right device with the BT service.



(b) When the Zenith inspection camera's *mise en place* service (left bowls) detects that the user has assembled the ingredients, pre-heating the oven will start. The Sequencer operator (SEQ) is used for that implementation. In the meantime, it can aid the user by having other services done in parallel via the Paralleliser operator (PAR) (e.g., weighing ingredients and mixing them).

Fig. 6: A BT IoT cooking workflow example of adding device interdependencies (Figure 6a) and its DX-MAN equivalent (Figure 6b).

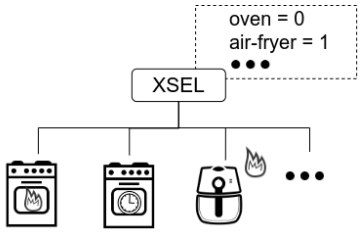


Fig. 7: The DX-MAN's exclusive selector is updated by the MAPE-K loop. In this example, the airfryer is chosen over the oven; subsequently, a workflow will be created based on the services provided by the airfryer.

$broil, heat = [200C], time = [30mins] >$ then $airfryer < heat = [180C], time = [15mins] >$ where mapping is done internally by some function $f_{airfryer}(oven\_variables)$ defined by a domain expert. To construct the IoT cooking workflow following DX-MAN entails having an exclusive selector operator which chooses the appropriate services of the preferred device to accomplish the recipe; we refer the reader to Figure 7. The equivalent implementation with BTs requires adding a fallback node with a condition (i.e., to check whether the user prefers the oven or the airfryer), similarly to Figure 3b. This process creates multiple internal nodes. Again, one can argue that BTs can be created dynamically, but the same argument stated above (in our discussion on Adding Device Interdependencies) also holds true in this use case. Thus, in the DX-MAN case the parameter complexity is somewhat more than the BT, since the exclusive selector needs to denote which devices will be used, whereas in the BT scenario there can be an overwhelming amount of execution complexity due to the introduction of multiple internal fallback nodes with conditions.

2) Block-Based Visual Programming Language: End-users should not be limited to only editing existing recipes but should be able to create new recipes from the ground up with their existing IoT devices. In this section, we analyse the construction of IoT cooking workflows with Block-Based programming languages. The transition from BTs to a Block-Based programming language was explored previously [51]. Below we discuss some block-based functionalities that, through additions and modifications, may create complicated underlying semantic model structures.

a) Multiple Conditions: Cooking can be quite versatile as presented earlier. For instance, different preferences from different users (e.g., a guest who prefers non-spicy options) can result in a daunting number of conditions. This leads to many different variations. In a Block-Based programming language, users can select numerous conditions (and loops) based on their preference. With its exclusive selector operator, DX-MAN minimises execution complexity but would increase parameter complexity. However, it can omit non-relevant parameters when presented to a user. In contrast, due to its condition leaf nodes and more internal nodes, the BT architecture adds more execution complexity; the same argument from Section IV-A1 holds here. Specifically, a large divergence in execution complexity is notable where the BT tends to scale vertically rather than horizontally, involving more conditions rather than actions, since the conditions can be set as parameters in DX-MAN, making it easier to track, subject to the visualisation medium.

b) Composing Services: The primary action in a Block-Based language is the invocation of specific services of various IoT devices. We referred earlier to this architectural attribute as compositionality (Section II-B3). This principle can be expanded to merge cooking workflows of different users, where each workflow details their own preferences and requirements. In BTs, there is no guarantee of compositionality. In fact, [52] examined the use of the Communicating sequential processes (CSP) language to model hierarchies and states in order to test the compositionality of given end-user requirements and parsing to and from BTs. In the DX-MAN model, compositionality is built upon algebraic semantics, and the composition operators receive multiple services as operands to construct spaces that contain multiple tasks/workflows (see Section III-B). In particular, the exclusive selector operator can be customised with a set of constraints to determine how the combination of services (DX-MAN's operations) can be executed.

c) Recoverability: A major hindrance to the adoption of smart environments is the hesitancy of users to trust them [20]. A step towards earning the user's trust is to set appropriate recovery options should their plan face an unexpected behaviour. BTs have the internal fallback node which prioritises a leaf action and, should that fail, proceeds to the next available action (see Section III-A). This makes it easy to track the available options if a service fails. In comparison, in the DX-MAN model, there is no fallback node. However, one can approach this issue by engineering the exclusive selector to prioritise a service to execute. These conditions can be updated via the MAPE-K loop (see Section III-B). Arguably the BTs provide a transparent approach to all the fallback options which aids its readability. Consequently, its execution complexity is high since it shows all the possible fallback actions. The DX-MAN model depends on its exclusive selector capability to make the "right" choices. It does not offer the transparency that BT does, hence decreasing its execution complexity. However, its parameter complexity is higher due to the number of conditions the exclusive selector may incorporate. In addition, its memory complexity can be higher if the end-user defines a set of parameters to then get updated/configured by the MAPE-K loop in order to accommodate the current environment (since the end-user will need to recall what parameters they set, to be configured for a new context).

V. Limitations & Future Work

In this paper, we have described a three-tier EUD framework, inspired by the meta-design framework where end-users with different levels of expertise participate in the development of a product—in this case, of IoT cooking workflows. A key characteristic of the framework is enabling non-technical end-users to create rules or workflows using the aforementioned EUD paradigms; these can then be analysed or augmented by the more technical kind of end-users in the form of either of two semantic representations: BTs and the DX-MAN model. We analysed these semantic representations using workflow complexity estimates. A limitation of this work, however, is the lack of analysis of the extent to which technical end-users appreciate the visualisation of either the BT or the DX-MAN model's notations and structure. In our future work, we aim to address this, as well as the question of how end-users can also develop their own IoT cooking workflows with these models. Moreover, the expressive power of these semantic representations will be addressed in our future work, by a possible goal-oriented design study involving human participants and a set of smart-kitchen devices to analyse and compare how "real" workflows (i.e. multiple recipes, users, devices, options) can be expressed in each. This can further expand the discussion on the ease of analysis of multiple overlapping workflows; this will also lead to insights on the choice of visual notations [53] and practices to reduce complexity. Finally, we will analyse other semantic representations, e.g., Petri Nets and task models, which have been used for ambient intelligent environments.

VI. Conclusion

The purpose of the current study is to compare two possible IoT cooking workflow representations (i.e., BTs and the DX-MAN model) for smart kitchen end-users. We introduced the notion of meta-design in EUD, which refers to the environment where there can be end-users with varying levels of domain expertise (i.e., from novices to professional chefs) and programming/development skills (i.e., end-users to technology developers). We investigated how end-users can visualise workflows and how the semantic representations can work with widely used EUD paradigms, namely, Event Condition Action and the Block-based programming language. To enable comparison, we discussed workflow complexities [18] and feasibility. We demonstrated that execution complexity tends to be higher with BTs, whereas parameter complexity tends to be somewhat higher in the DX-MAN model. The DX-MAN model benefits from having an in-built algebraic composition mechanism that can combine services from different IoT devices into workflows which, in turn, can be hierarchically combined with other workflows. This notion makes it straightforward for a EUD interface to combine services. BTs, however, benefit from its fallback node which can act as an immediate step to recover from an unexpected scenario. Here, the DX-MAN model would instead need to re-iterate via its MAPE-K loop and update parameters of exclusive selectors to accomplish this, which is more complicated parameter-wise. Lastly, the DX-MAN model is advantageous in terms of visualisation since its representations are more succinct (which minimise the execution complexity). Furthermore, its workflows are always traversed in the same direction (i.e., left to right), unlike BTs, which can be beneficial for downstream visualisation.

References

[1] F. Paternò and C. Santoro, *A Design Space for End User Development in the Time of the Internet of Things*. Cham: Springer International Publishing, 2017, pp. 43–59.

[2] J. Coutaz and J. L. Crowley, "A first-person experience with end-user development for smart homes," *IEEE Pervasive Computing*, vol. 15, no. 2, pp. 26–39, 2016.

[3] B. Ur, E. McManus, M. Pak Yong Ho, and M. L. Littman, "Practical trigger-action programming in the smart home," in *Proc. of the SIGCHI conf. on Human Factors in Computing Systems*, ser. CHI '14. ACM, 2014, p. 803–812.

[4] G. Ghiani, M. Manca, F. Paternò, and C. Santoro, "Personalization of context-dependent applications through trigger-action rules," *ACM Trans. Comput.-Hum. Interact.*, vol. 24, no. 2, Apr. 2017.

[5] Y. Francillette, S. Gaboury, A. Bouzouane, and B. Bouchard, "Towards an adaptation model for smart homes," in *Inclusive Smart Cities and Digital Health*, C. K. Chang, L. Chiari, Y. Cao, H. Jin, M. Mokhtari, and H. Aloulou, Eds. Cham: Springer International Publishing, 2016, pp. 83–94.

[6] "The next generation 1996 lexicon a to z: Npc (nonplayer character)," p. 38, March 1996.

[7] Y. A. Sekhavat, "Behavior trees for computer games," *International Journal on Artificial Intelligence Tools*, vol. 26, no. 02, p. 1730001, 2017.

[8] M. Colledanchise and P. Ögren, "Behavior trees in robotics and ai: An introduction," *ArXiv*, vol. abs/1709.00084, 2017.

[9] D. Arellanes and K.-K. Lau, "Evaluating iot service composition mechanisms for the scalability of iot systems," *Future Generation Computer Systems*, vol. 108, pp. 827–848, 2020.

[10] ——, "Algebraic service composition for user-centric iot applications," in *Internet of Things – ICIOT 2018*, D. Georgakopoulos and L.-J. Zhang, Eds. Cham: Springer International Publishing, 2018, pp. 56–69.

[11] ——, "Workflow variability for autonomic iot systems," in *2019 IEEE International conf. on Autonomic Computing (ICAC)*, 2019, pp. 24–30.

[12] Z. Xing, Z. Hong, and L. Yulong, "A petri-net based context-aware workflow system for smart home," in *2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops PhD Forum*, 2012, pp. 2336–2342.

[13] B. D. Carolis, S. Ferilli, and D. Redavid, "Incremental learning of daily routines as workflows in a smart home environment," *ACM Trans. Interact. Intell. Syst.*, vol. 4, no. 4, Jan. 2015.

[14] R. Martinho, D. Domingos, and A. Respício, "Evaluating the reliability of ambient-assisted living business processes," in *Proc. of the 18th International conf. on Enterprise Information Systems*, ser. ICEIS 2016. Setubal, PRT: SCITEPRESS - Science and Technology Publications, Lda, 2016, p. 528–536.

[15] B. Avenoğlu and P. E. Eren, "A context-aware and workflow-based framework for pervasive environments," *Journal of Ambient Intelligence and Humanized Computing*, vol. 10, no. 1, pp. 215–237, Jan 2019.

[16] S. R. Humayoun, Y. Dubinsky, and R. AlTarawneh, "Using iotgolog to formalize iot scenarios," in *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, 2015, pp. 234–238.

[17] B. Margaret and S. Christopher, *The Encyclopedia of Human-Computer Interaction*, 2nd ed. Interaction Design Foundation, 2013, ch. 10.

[18] A. Keller, A. B. Brown, and J. L. Hellerstein, "A configuration complexity model and its application to a change management system," *IEEE Transactions on Network and Service Management*, vol. 4, no. 1, pp. 13–27, 2007.

[19] S. Kerr, O. Tan, and J. Chua, "Cooking personas: Goal-directed design requirements in the kitchen," *International Journal of Human-Computer Studies*, vol. 72, p. 255–274, 02 2014.

[20] A. Coskun, G. Kaner, and I. Bostan, "Is smart home a necessity or a fantasy for the mainstream user? a study on users' expectations of smart household appliances," *International Journal of Design*, vol. 12, 04 2018.

[21] K. Czarnecki, "Variability in software: State of the art and future directions," in *Fundamental Approaches to Software Engineering*, V. Cortellessa and D. Varró, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 1–5.

[22] D. Arellanes and K.-K. Lau, "Exogenous connectors for hierarchical service composition," in *2017 IEEE 10th conf. on Service-Oriented Computing and Applications (SOCA)*, 2017, pp. 125–132.

[23] A. Schmidt, "Programming ubiquitous computing environments," in *End-User Development*, P. Díaz, V. Pipek, C. Ardito, C. Jensen, I. Aedo, and A. Boden, Eds. Cham: Springer International Publishing, 2015, pp. 3–6.

[24] C.-Y. Chang, D.-A. Huang, D. Xu, E. Adeli, L. Fei-Fei, and J. C. Niebles, "Procedure planning in instructional videos," in *Computer Vision – ECCV 2020*, A. Vedaldi, H. Bischof, T. Brox, and J.-M. Frahm, Eds. Cham: Springer International Publishing, 2020, pp. 334–350.

[25] R. Maskeliūnas, R. Damaševičius, and S. Segal, "A review of internet of things technologies for ambient assisted living environments," *Future Internet*, vol. 11, no. 12, 2019.

[26] B. O'Boyle, "Best smart ovens 2021: The tech changing the way we cook," Feb 2021. [Online]. Available: https://www.pocket-lint.com/smart-home/buyers-guides/146457-best-smart-ovens

[27] A. Neumann, C. Elbrechter, N. Pfeiffer-Leßmann, R. Kõiva, B. Carlmeyer, S. Rüther, M. Schade, A. Ückermann, S. Wachsmuth, and H. J. Ritter, "Kognichef: A Cognitive Cooking Assistant," *KI - Künstliche Intelligenz*, vol. 31, no. 3, pp. 273–281, Aug. 2017.

[28] S. Reichel, T. Muller, O. Stamm, F. Groh, B. Wiedersheim, and M. Weber, "MAMPF: An Intelligent Cooking Agent for Zoneless Stoves," in *2011 Seventh International conf. on Intelligent Environments*, Jul. 2011, pp. 171–178.

[29] P. Olivier, G. Xu, A. Monk, and J. Hoey, "Ambient kitchen: Designing situated services using a high fidelity prototyping environment," in *Proc. of the 2nd International conf. on Pervasive Technologies Related to Assistive Environments*, ser. PETRA '09. New York, NY, USA: Association for Computing Machinery, 2009.

[30] I. Georgievski and M. Aiello, "Automated planning for ubiquitous computing," *ACM Comput. Surv.*, vol. 49, no. 4, Dec. 2016.

[31] G. Fischer and T. Herrmann, "Socio-technical systems: A meta-design perspective." *IJSKD*, vol. 3, pp. 1–33, 01 2011.

[32] G. Fischer, "Symmetry of ignorance, social creativity, and meta-design," *Knowledge-Based Systems*, vol. 13, no. 7, pp. 527–537, 2000.

[33] L. Lessig, *Remix making art and commerce thrive in the hybrid economy*. London: Bloomsbury, 2008.

[34] B. A. Chagas, D. F. Redmiles, and C. S. de Souza, "End-user development for the internet of things or how can a (smart) light bulb be so complicated?" in *2017 IEEE Symposium on Visual Languages and Human-Centric Computing*, Oct 2017, pp. 273–277.

[35] F. Cabitza, D. Fogli, R. Lanzilotti, and A. Piccinno, "End-user development in ambient intelligence: A user study," in *Proc. of the 11th Biannual conf. on Italian SIGCHI Chapter*, ser. CHItaly 2015. New York, NY, USA: ACM, 2015, p. 146–153.

[36] G. Desolda, C. Ardito, and M. Matera, "Empowering end users to customize their smart environments: Model, composition paradigms, and domain-specific tools," *ACM Trans. Comput.-Hum. Interact.*, vol. 24, no. 2, Apr. 2017.

[37] B. Cheng, M. Wang, S. Zhao, Z. Zhai, D. Zhu, and J. Chen, "Situation-aware dynamic service coordination in an iot environment," *IEEE/ACM Trans. Netw.*, vol. 25, no. 4, p. 2082–2095, Aug. 2017.

[38] M. Manca, Fabio, Paternò, C. Santoro, and L. Corcella, "Supporting end-user debugging of trigger-action rules for iot applications," *International Journal of Human-Computer Studies*, vol. 123, pp. 56 – 69, 2019.

[39] F. Corno, L. De Russis, and A. Monge Roffarello, "My iot puzzle: Debugging if-then rules through the jigsaw metaphor," in *End-User Development*, A. Malizia, S. Valtolina, A. Morch, A. Serrano, and A. Stratton, Eds. Cham: Springer International Publishing, 2019, pp. 18–33.

[40] M. A. Kuhail, S. Farooq, R. Hammad, and M. Bahja, "Characterizing visual programming approaches for end-user developers: A systematic review," *IEEE Access*, vol. 9, pp. 14 181–14 202, 2021.

[41] D. Rough and A. Quigley, "Jeeves - a visual programming environment for mobile experience sampling," in *2015 IEEE Symposium on Visual Languages and Human-Centric Computing*, Oct 2015, pp. 121–129.

[42] B. Broll, Ákos Lédeczi, H. Zare, D. N. Do, J. Sallai, P. Völgyesi, M. Maróti, L. Brown, and C. Vanags, "A visual programming environment for introducing distributed computing to secondary education," *Journal of Parallel and Distributed Computing*, vol. 118, pp. 189–200, 2018.

[43] F. Paternò and C. Santoro, "End-user development for personalizing applications, things, and robots," *International Journal of Human-Computer Studies*, vol. 131, pp. 120–130, 2019.

[44] A. Johansson and P. Dell'Acqua, "Comparing behavior trees and emotional behavior networks for npcs," in *2012 17th International conf. on Computer Games*, 2012, pp. 253–260.

[45] E. Coronado, F. Mastrogiovanni, and G. Venture, "Development of intelligent behaviors for social robots via user-friendly and modular programming tools," in *2018 IEEE Workshop on Advanced Robotics and its Social Impacts*, 2018, pp. 62–68.

[46] C. Paxton, A. Hundt, F. Jonathan, K. Guerin, and G. D. Hager, "Costar: Instructing collaborative robots with behavior trees and vision," in *IEEE International conf. on Robotics and Automation*, 2017, pp. 564–571.

[47] J. Kephart and D. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[48] H. Zuse, "Properties of software measures," *Software Quality Journal*, vol. 1, no. 4, pp. 225–260, Dec 1992.

[49] M. Herschel, R. Diestelkämper, and H. Ben Lahmar, "A survey on provenance: What for? what form? what from?" *The VLDB Journal*, vol. 26, no. 6, p. 881–906, Dec. 2017.

[50] H. Stitz, S. Luger, M. Streit, and N. Gehlenborg, "Avocado: Visualization of workflow–derived data provenance for reproducible biomedical research," *Computer graphics forum : journal of the European Association for Computer Graphics*, vol. 35, pp. 481 – 490, 2016.

[51] E. Coronado, F. Mastrogiovanni, and G. Venture, "Development of intelligent behaviors for social robots via user-friendly and modular programming tools," in *2018 IEEE Workshop on Advanced Robotics and its Social Impacts*, 2018, pp. 62–68.

[52] R. J. Colvin and I. J. Hayes, "A semantics for behavior trees using csp with specification commands," *Science of Computer Programming*, vol. 76, no. 10, pp. 891–914, 2011, integrated Formal Methods.

[53] D. Moody, "The "physics" of notations: Toward a scientific basis for constructing visual notations in software engineering," *IEEE Transactions on Software Engineering*, vol. 35, no. 6, pp. 756–779, 2009.