Journal of Computer Languages 58 (2020) 100970

Contents lists available at ScienceDirect

Journal of Computer Languages

journal homepage: www.elsevier.com/locate/cola

Visual Programming Environments for End-User Development of intelligent and social robots, a systematic review

Enrique Coronado[a,*], Fulvio Mastrogiovanni[b], Bipin Indurkhya[c], Gentiane Venture[a]

[a] Department of Mechanical Systems Engineering, Tokyo University of Agriculture and Technology, Tokyo, Japan
[b] Department of Informatics, Bioengineering, Robotics and Systems Engineering, University of Genoa, Genoa, Italy
[c] Department of Cognitive Science, Institute of Philosophy, Jagiellonian University, Cracow, Poland

ARTICLE INFO

Article history:
Received 9 September 2019
Revised 28 February 2020
Accepted 18 April 2020
Available online 20 May 2020

2019 MSC:
00-01
99-00

Keywords:
Visual Programming Environment
End-User Development
Human–robot interaction
Social robot
Robotics

ABSTRACT

Robots are becoming interactive and robust enough to be adopted outside laboratories and in industrial scenarios as well as interacting with humans in social activities. However, the design of engaging robot-based applications requires the availability of usable, flexible and accessible development frameworks, which can be adopted and mastered by researchers and practitioners in social sciences and adult end users as a whole. This paper surveys *Visual Programming Environments* aimed at enabling a paradigm fostering the so-called *End-User Development* of applications involving robots with social capabilities. The focus of this article is on those *Visual Programming Environments* that are designed to support social research goals as well as to cater for professional needs of people not trained in more traditional text-based computer programming languages. This survey excludes interfaces aimed at supporting expert programmers, at allowing industrial robots to perform typical industrial tasks (such as pick and place operations), and at teaching children how to code. After having performed a systematic search, sixteen programming environments have been included in this survey. Our goal is two-fold: first, to present these software tools with their technical features and *Authoring Artificial Intelligence* modeling approaches, and second, to present open challenges in the development of *Visual Programming Environments* for end users and social researchers, which can be informative and valuable to the community. The results show that the most recent such tools are adopting distributed and Component-Based Software Engineering approaches and web technologies. However, few of them have been designed to enable the independence of end users from high-tech scribes. Moreover, findings indicate the need for (i) more objective and comparative evaluations, as well as usability and user experience studies with real end users; and (ii) validations of these tools for designing applications aimed at working "in-the-wild" rather than only in laboratories and structured settings.

1. Introduction

Robots are programmable machines that are nowadays commonly available and used in universities, research institutes, and manufacturing industries. Traditionally, they have been used to perform high-speed, efficient and repetitive tasks in hazardous and industrial environments, often requiring few or no interactive capabilities [1,2]. These cases differ significantly from social robots, for which the main goals are to play useful social roles and engage different types of users through meaningful, natural, suitable, and safe interactions [3,4]. Despite being one of the most relevant emergent technologies according to the World Economic Forum [5], the successful adoption and acceptance of social robots into our society still requires many challenges to be solved [6,7]. Similar to the early years of computing hardware, current social robotics applications and research are widely dominated by high-tech scribes, i.e., experts in programming or engineers [8]. Moreover, these applications are generally designed, executed and evaluated in structured, *closed* and controlled environments, such as laboratories, and under the supervision of experts in robotics engineering [9]. However, social robots are aimed at being used by the general public and at performing "in-the-wild", i.e., in unstructured, dynamic, open and everyday environments [9].

In order to successfully perform in these scenarios, social robots must be able to provide useful, safe, usable, valuable, enjoyable and meaningful experiences [7,10]. Unlike the requirements posed by the introduction of robots in industrial scenarios, use cases in which robots must interact with people using social norms and

* Corresponding author.
E-mail address: enriquecoronadozu@gmail.com (E. Coronado).

https://doi.org/10.1016/j.cola.2020.100970
2590-1184/

conventions are better approached by UX/UI designers, researchers, practitioners in social sciences. Therefore, the inclusion of this new type of users in the design process of social robots is keystone to ensure the successful performance of social robot based applications. However, people belonging to this category are traditionally skillful in domains profoundly different from advanced robot and software development, and oftentimes lack the required level of expertise in advanced engineering topics, which are typically needed to implement complex robot behaviors. Examples of approaches enabling the inclusion of non-roboticists in the creation of interactive applications are user-centered [11] and participatory design [12]. Recent examples in Social Robotics applying these methods are presented in [13,14]. As described in [15], user-centered and participatory design endorse the "design for use before use" paradigm, which requires a clear division of labour between the people assigned for the creation of applications at design time and the people able to use and redesign the application at run time. According to [16], approaches requiring this division of labour have become problematic in many software development areas due to: (i) a lack of expert software developers or (manpower) able to grasp and attend all possible users as well as their needs; (ii) the dynamic change of requirements, which are often specific to individual domain applications; and (iii) possible misunderstandings between expert software developers and their users due to the difference in backgrounds and practices.

Recently, End-User Development (EUD) has emerged as a suitable alternative to those approaches requiring a division of labour [16]. This is done by enabling novice users of computers and people without training on traditional programming languages, who are often denoted as end users, to redesign their own applications not only at design time but also at run time [17]. The goal of EUD is to evolve from *easy-to-use* to *easy-to-develop* interactive technologies [16]. Such a goal is not limited to software but can also include hardware artifacts, such as those manufactured using 3D printing technology [16]. A new and broad definition based in the meta-design manifesto [18] also considers EUD as a socio-technical activity whereby users can develop all software and hardware systems that they use in their everyday life [19], therefore enabling independence of the owners of the problems, i.e., end users themselves, apart from the high-tech scribes [8]. The concept of EUD is related to End-User Programming (EUP) and End-User Software Engineering (EUSE). On the one hand, EUP is often considered as a subset of EUD [20], because it focuses only on the techniques used to enable end users to write their own programs, such as visual programming, domain-specific languages and natural language programming. In contrast, EUD not only focuses on the program creation phases but also on the methods and tools that can support the entire software development life-cycle [17]. This requires reaching independence from high-tech scribes during the use, redesign, configuration, and extension of the software and hardware artifacts [19]. On the other hand, EUSE takes a different approach compared to EUP and EUD. This is because EUSE mostly focuses on providing end users with solutions derived from traditional software engineering, such as debugging and version control, to promote the creation of high-quality software that is reusable, reliable and efficient [20,21].

In the past few years, a number of systems have been proposed to tackle EUD challenges for robotics systems at different levels, e.g., motion planning and execution frameworks adopting *Programming by Demonstration* (PbD) [22,23], or the use of *Natural Language Processing* (NLP) to provide robots with instructions about how to carry out a certain task [24]. However, *Visual Programming Environments* (VPE) still follow the EUP and EUD approach, offering the best trade-off between usability (being easy to learn and easy to use) and the overall *complexity* characterizing the robot-based behaviors that can be developed with these tools. VPEs integrate a selected *Visual Programming Language* (VPL) to enable their users to create applications on the basis of such graphical elements as icons, blocks, arrows, forms, and figures, among others, rather than code only [25,26]. The relevance of these development tools has recently increased not only in Robotics-related use cases, but also in other fields of Computer Science such as the *Internet of Things* (IoT) [25,27], video game development [28,29], mobile application development [30], and Virtual/Augmented Reality [31]. Due to their aforementioned flexibility and relevance, this article focuses on the EUD tools using VPEs for the design of Social Robotics applications.

As described by Barricelli et al. [17], most relevant literature reviews on EUD, such as [32–34], present a limited number of approaches and techniques. These relatively old literature review articles also tend to omit applications in Robotics. Moreover, the reviews presented in such articles have been performed and analyzed in the research domain of their authors [17], which differ from Robotics. Two more recent systematic reviews on EUD approaches are [35] and [17]. The authors of [35] conducted a 10-year (2007–2017) systematic search, in which 21 articles were selected. However, none of them belongs to the Robotics field. In contrast, Barricelli et al. [17] presents an overview of applications, tools and techniques in EUD, EUP and EUSE over 17 years (2000–2017). From the 165 papers selected in [17], only four are in the Robotics area, out of which only two are relevant for the focus of this article. Finally, a very recent narrative review of EUD-inspired software applied to domains such as smart homes, industrial and humanoid Robotics, task automation, and applications for human assistance is presented in [36]. However, the review proposed in [36] mainly focuses the analysis and proposed challenges to IoT-specific approaches and rule-based systems, such as those based on the trigger-action paradigm [37]. In fact, most literature reviews on EUD tend to omit in their analysis the more recent tools, technologies, and approaches used in Robotics aimed at enabling the development of more advanced, reactive and robust robot systems. Relevant methods often omitted are related to the behavior modeling approaches enabling end users to create intelligent and social robots. These approaches, which are often denoted as *Authoring Artificial Intelligence* (AAI) or simply Artificial Intelligence (AI) architectures [38], enable the control of processes in which intelligent *agents* can evaluate their environment to perform decision making. These AAI-based methods are nowadays widely used in areas in which the development of complex and intelligent physical and virtual agents is needed, such as Robotics or game development [39]. Three of the main research goals of AAI-based methods are: (i) overcoming limitations related to modularity, reliability, reusability, and robustness presented by the classical agent behavior modeling methods, such as rule-based systems and scripting [38]; (ii) enabling their use for EUD and interaction design tasks [39]; and (iii) generating more intelligent, reactive, believable, suitable, and explainable agent behaviors [40]. Due to this, the role played by the AAI-based methods in EUD and EUP must not be omitted. Moreover, current systematic and narrative reviews in EUD also tend to omit in their analysis those relevant development practices and approaches nowadays used to create more advanced, reusable, scalable, interactive and reliable robot systems, and in particular, the use of frameworks supporting Component-based Software Engineering (CBSE) for Robotics. This approach has become quasi-standard for the development of software architectures for robots [41,42]. Therefore, their analysis is fundamental for a better understanding of how to develop more advanced, usable and robust software back-ending EUD and EUP for Robotics. Other recent reviews describing interesting intuitive programming tools, such as [43,44], focus on educational contexts for children rather than EUD applications for adult users.

The limitations of the aforementioned studies indicate that there is a need for a more in-depth systematic identification of re-

Table 1
Cognitive dimension definitions.

Dimension	*Definition*
Abstraction gradient	Types and availability of abstraction mechanisms
Closeness of mapping	Closeness of representation to domain
Hidden dependencies	Important links between entities are not visible
Premature commitment	Constraints on the order of doing things
Viscosity	Resistance to change
Visibility	Ability to view components easily
Diffuseness	Verbosity of language
Error-proneness	Notation invites mistakes
Hard mental operations	High demand on cognitive resources
Progressive evaluation	Work-to-date can be checked at any time
Provisionality	Degree of commitment to actions or marks
Role-expressiveness	The purpose of a component is readily inferred
Secondary notation	Extra information in means other than formal syntax
Consistency	Similar semantics are expressed in similar syntactic forms

search articles enabling EUD for Social Robotics, with the aim of providing a broader overview and understanding of the development approaches, tools, and practices enabling end users to create intelligent and interactive applications with social robots. Towards this end, this article presents a systematic search and analysis [45] of VPEs aimed at enabling the EUD paradigm for social human–robot interaction (HRI) and everyday life applications. The main contribution of this paper is as follows. By systematically identifying and analyzing relevant VPEs for EUD of application with social robots, we provide: (i) a more complete overview of current scenarios for EUD and EUP solutions in this area; (ii) an in-depth analysis of the algorithms and modeling approaches currently used to enable the creation of more intelligent and robust agents; (iii) a presentation of the trends, practices, and technologies used for the development of VPEs for social robots; and (iv) the definition of open challenges and future directions specific to Social Robotics.

The organization of this paper is as follows. Section briefly presents basic concepts used in this article for the analysis of VPEs. Section 3 describes the main concepts underlying VPE-based design, with an emphasis on Robotics-related requirements. Section 4 describes the methodology applied to perform a systematic analysis of the literature. Section 5 presents the VPE tools found in the literature following our methodology. Section 6 discusses behavior modeling approaches used in VPEs for developing intelligent social robots. Section 7 describes more relevant software tools and technologies executing at the back-end of these VPEs. Section 8 presents relevant open challenges, thereby proposing a road-map for further research directions. Conclusions follow.

2. Appropriate abstraction level for programming social robots

The concept of *cognitive dimension* is a framework used to analyze complex software tools such as programming languages and interactive user interfaces [46,47]. It can be used to identify usability problems in the early stages of the design of a user interface and to perform iterative design. Brief definitions, based in [48] and [49], of the most relevant cognitive dimensions are shown in Table 1. As a usability principle, design of programming tools based in cognitive dimensions must deal with a set of trade-offs, i.e., attempts to improve any dimension always affects other dimensions. Therefore, cognitive dimension design must be goal-oriented by selecting the dimensions that are more important for the target audience.

Recently, the work in [50] studied cognitive dimension and usability trade-offs when considering the programming of social robots. This analysis resulted in a proposal for a robot programming model that decomposes the social and intelligent abilities of robots in five abstraction levels, namely hardware primitives, algorithms primitives, social primitives, emergent primitives, and methods for controlling primitives. In this model, the lowest abstraction level is the hardware primitives that allow programmers to retrieve sensory information from hardware devices and control robot inputs, e.g., LEDs and motors. The second abstraction level is the algorithm primitives that are used to build low-level interactive, perceptual and control capabilities in social robots, e.g., face tracking, sound source localization, and inverse kinematics. The third level is that of social primitives, which contains intuitive and reusable social interactive capabilities that are close to the domain expertise of the general end users. At the fourth level, emergent primitives are built from a combination of social primitives, e.g., gaze, speech, and gestures, to create high-level social behaviors, such as those related to emotion-inducing behaviors. Finally, the fifth level contains the control primitives that are in charge of performing decision making based on the current status of the interaction. The simplest way of doing this task is by using *if-then-else* rules. A description of the methods used in EUD tools for Social Robotics for controlling primitives is the main focus of Section 6. Findings of [50] suggest that using too many low-level abstractions, i.e., hardware and algorithm primitives, for the development of programming tools for social robots negatively affects their usability. Such low-level primitives tend to require hard mental operations and produce error-prone notations. At the same time, using too many emergent primitives affects the viscosity positively, but the expressive power of the programming tools and hidden dependencies negatively. In order to reach good usability and cognitive dimension trade-offs for the end user, these programming tools must use as many social primitives as possible. A deeper description of these primitives and their influence on the usability and cognitive dimension trade-offs of a programming tool for social robots are explained in-depth in [50].

3. Visual programming environments in robotics

With some simplifications, three main Robotics-related scenarios can be identified where VPEs play a decisive role, namely (i) industry settings, (ii) science, technology, engineering and mathematics (STEM) education, and (iii) end-user applications. Two major factors form a basis for this classification, namely the required programming abstraction level [50] and the target users.

One key feature that distinguishes among these classes of use cases is based on the most appropriate level of abstraction characterizing the programming *operators* and *primitives*, i.e., the available visual elements, which can provide the required usability, intuitiveness, ease of learning, and flexibility. While adopting low-level programming abstraction primitives can enhance the flexibility and the level of code reuse associated with VPEs, it also decreases their usability and intuitiveness [50]. Furthermore, the capability associated with easy-to-learn approaches can be negatively affected when developers must deal with a *mixture* of unbalanced abstraction levels [47,50].

A second major difference is the target end users population in its own right. In use cases grounded in industry scenarios, the use of VPE-based approaches is geared towards reducing the costs associated with the development and maintenance of robot applications on the shop-floor or in the manufacturing cell by human operators who are new or untrained in programming [51]. However, users of industry-oriented VPEs still require some exper-

tise in low-level programming and Robotics notations. Examples of low-level notations presented in some VPEs for industrial settings are coordinate frames, tools, materials, joint velocities, end-effector orientations and positions, and hardware configurations [52–54]. In general, industry-oriented VPEs are mainly focused on enabling robots to execute a set of well-defined sequences of repetitive and accurate tasks, e.g., assembly, pick-and-place, welding and material handling [52], rather than enabling them to play complex and diverse social roles, e.g., teacher, friend or companion. Some advanced VPEs for industrial settings also enable a mixed approach with PbD methods [52,53].

Use cases related to the adoption of VPEs in STEM and – in general – educational settings are typically aimed at children or new learners of general-purpose programming languages for developing *toy* programs rather than real-world applications. This type of VPEs is characterized by two main peculiarities. Firstly, they must be based on suitable approaches to enforce learning Computer Science or Robotics-related topics, such as the management of sensors or actuators, coding, functions, data structures, or algorithms. Secondly, these VPEs must be engaging and sufficiently easy-to-use to keep students interested and motivated during programming sessions. The abstraction level typically encoded in this type of interfaces depends also on the age of the target learners [55]. For elementary and middle school students, these software development environments must favour simplicity, intuitiveness and avoid the intrinsic complexity of general-purpose programming languages [56]. However, environments for students in high school and above often require the use of low-level general-purpose programming syntax, e.g., conditionals, loops and functions, to enable an easy transfer of knowledge to general-purpose programming languages or more complex approaches in advanced courses [57]. Nowadays, new STEM-targeted educational VPEs coupled with robot toys appear every year, notable examples being the interfaces for such robots as Cozmo [58] and Thymio [59].

Programming robots and learning how to use the available VPEs in both industrial and STEM-related educational scenarios are generally the main tasks or objectives of their target users. As mentioned above, these tasks require some low-level expertise in Robotics (in the case of industrial settings) or the user needs to acquire a complex body of knowledge by time-consuming training processes (in the case of STEM-related educational scenarios) [55]. This greatly differs from what is postulated by EUD approaches for domain-specific users [34,60], whereby *programming* is seen as an optional task to support work activities carried out by an end user rather than being a main learning or work objective. This may be because many domain-specific end users do not have the time and motivation to learn how to use low-level Robotics software frameworks [61,62]. Therefore, VPEs for end users require more intuitive interfaces, mainly based on programming notations that are close to the domain knowledge of the general user. In many cases, these VPEs must also be flexible enough to enable the creation of complex, dynamic and engaging social interactive experiences with robots. Such interactions often require the use of multimodal approaches, e.g., gesture and speech recognition, expression of emotions, and engaging dialogues, among others [63]. Examples of domain-specific end users are teachers developing robot tutors and helpers, artists programming a choreography or defining a script for robot-related artistic performances [64], sellers creating interactive experiences for customers [65], or therapists using robots to help in therapy sessions, just to name a few [62].

This article is targeted to discuss VPEs suitable to enable the adoption of EUD-based paradigms for the creation of social interaction applications by domain-specific end users.

According to the discussion in [66], most common VPL approaches can be categorized as: (i) form-filling, (ii) data-flow, and (iii) block programming.

Form-filling VPLs generally require the use of standard input forms, such as buttons and checkboxes, along with images to guide the user step-by-step. A popular AAI approach used for modeling robot behaviors in this type of VPLs is the use of trigger-action rules [27,66]. While such VPLs are popular in different IoT environments, such as smart homes [66], they are very poorly explored in Robotics [67]. This can be due to the widely known limitations that these approaches present when they are required for producing intelligent agent behaviors [39,40]. Moreover, a lack of structure when defining disjoint *ad hoc* rules can make trigger-action systems unstable and error-prone when creating relatively complex programs, which requires the integration of additional tools for solving conflicts between rules [40,68,69]. Therefore, areas requiring the creation of more complex behaviors, such as video games and Robotics, prefer the use of more structured, robust, and expressive AAI-based approaches [38,70].

Data-flow is a commonly adopted VPL approach in Robotics, not only for creating EUD-based environments for non-technically skilled people but also for expert use in complex and robust applications, as described in [71]. A *data-flow* programming environment is represented using directed graphs [72]. Nodes in *data-flow* interfaces are referred to by different terms by various authors, such as *blocks, functions, icons, states, procedures*, and *boxes*. Nodes are connected by means of graphical *lines* (also referred to as *wires* or *arcs*), which represent the flow of data between functions/blocks or transitions between states.

Block programming, based on the *primitive-as-puzzle-piece* metaphor (also known as block-based visual programming) [57,73,74], is a recently adopted approach that is gradually gaining attention in the development of EUD-based interfaces [73]. Unlike *data-flow* tools, visual elements in block-based VPLs are not connected using lines. Instead, block-based VPLs programs are built by assembling jigsaw puzzle pieces, which present visual cues that indicate to the user how visual elements may be used. This makes a block-based VPL an intuitive and engaging approach that is able to stimulate user creativity [57,66]. Following this definition, we consider in our review those VPEs using popular block-based VPLs such as Scratch [75], Snap! [76] or Google Blockly [77,78].

4. Methodology

We follow the guidelines proposed in [45,79] for performing systematic reviews of Software Engineering papers. Systematic reviews are objective literature review studies used to identify relevant research papers, trends, gaps and challenges in some specific research area as well as to help in the position of research directions and activities [45]. Our protocol for performing a systematic review is based on recent and relevant systematic reviews with similar objectives and domain areas. Specifically, we follow [80], which focuses on the area of human–robot interaction, and [20], which focuses on the area of EUD. As described in [45,79], and applied in [80], the process involved in a systematic review consists of five parts: (S1) definition of the *review protocol*, whereby the research questions are careful defined as well as the methods used to answer them; (S2) definition of the *search strategy*, which aims at identifying the relevant research articles in the field; (S3) *documentation of the search process*, whereby readers are able to evaluate how completely and rigorously the search process has been performed; (S4) specification of *inclusion* and *exclusion* criteria, which are used to select core articles in the field; and (S5) a report of relevant data or information from each research article or software tool.

4.1. Research questions

One of the main focuses of this article is to complement recent literature review articles in EUD, such as [20] and [36], by

Table 2
Dimensions used to obtain general information of VPEs.

Label	*Dimension*	*Description and goal*
RQ1-D1	Name	Used to identify each tool analyzed in this article
RQ1-D2	EUP approach	Aims at discovering the VPEs technique used to enable the creation of end-user programs values in this dimension can be *form-filling, data-flow* or *block-based*
RQ1-D3	Target users	Aims at identifying the type of end users these VPEs have been designed for
RQ1-D4	Application domain	Aims at discovering the application domains in which these VPEs have been used to support end-users goals and needs
RQ1-D5	Target robot	Aims at identifying the type of robots supporting these VPEs

identifying and analyzing relevant tools and technologies integrating VPEs for enabling EUD and EUP in Social Robotics. The identification of these tools and technologies can be used to better understand the current scenario of EUD solutions for Social Robotics. We formulated the following research questions.

(RQ1) What VPE tools for Social Robotics have been proposed in the literature to support end-user research goals or professional needs? The dimensions used to respond to RQ1 are shown in Table 2. We propose RQ1-D1 (Name) to identify each tool resulting from the systematic search process. This enables a comparative analysis of these VPEs in the other research questions. Values of RQ1-D2 (EUP approach) are defined based on the VPEs classification presented in Section 3. The formulation of dimensions RQ1-D3 and RQ1-D4 is based on those proposed in [20] to obtain general information of EUD, EUP, and EUSE tools. Therefore, RQ1-D3 and RQ1-D4 are focused on identifying the main target users and applications of these tools, respectively. Finally, RQ1-D5 is used to identify the social and service robots supported by these VPEs. We propose these dimensions in order to get a general overview of the goals of relevant and recent VPEs for Social Robotics.

(RQ2) What robot behavior modeling AAI-based approaches have been used in these VPEs to enable the creation of intelligent social robots? RQ2 mostly focuses on: (i) how end users can effectively and intuitively compose programming primitives for the creation of their desired applications, and (ii) the methods enabling the control of these primitives. AAI-based approaches can be considered as those AI methods enabling the modeling and control of programming primitives used in VPEs for Social Robotics [50]. On the one hand, AAI-based approaches for social robots must be flexible and expressive enough, thereby providing end users with an ability to create interesting and complex behaviors. On the other hand, they must be intuitive and simple enough to allow for an easy creation and reuse of desired robot behaviors. Therefore, the main goal of RQ2 is to identify the advantages and disadvantages of different AAI-based methods and how they have been used to enable the modeling intelligence on social robots in VPEs. Dimensions proposed to answer this research questions are aimed at identifying the used AAI-based approaches in VPEs for Robotics (RQ2-D1) and the type of programming primitives generally used in these VPEs (RQ2-D2) (Table 3).

(RQ3) What technologies, evaluation methods, and software tools have been used by the authors of these frameworks? The focus of RQ3 is on the capabilities of the proposed tools to enable the independence of end users from high-tech scribes by supporting them in the entire life-cycle, and not only in the creation phase. For this, EUD tools must be accessible, easy-to-use and install, support end-user devices, and allow for an easy extension of software artifacts, e.g., the addition of perceptual capabilities or re-use with other virtual or physical agents. Dimensions used to answer RQ3 are summarized in Table 4. Dimensions RQ3-D1 and RQ3-D2 are proposed

Table 3
Dimensions used to answer RQ2.

Label	*Dimension*	*Description and goal*
RQ2-D1	AAI approach	Aims at identifying the type of agent behavior modeling approach used for controlling programming primitives on reviewed VPEs
RQ2-D2	Programming primitives	Aims to identify the type of programming primitives used in reviewed VPEs

Table 4
Dimensions used to answer RQ3.

Label	*Dimension*	*Description and goal*
RQ3-D1	Communication of modules	Aims at discovering whether these VPEs have been developed using good practices for the integration of isolated software modules or nodes
RQ3-D2	Software technologies	Aims to discover if these VPEs have been developed using modern technologies
RQ3-D3	Accessibility	Ascertains whether these VPEs are available online
RQ3-D4	Operating Systems (OS)	Aims at determining the degree of support that VPEs have for the OS used by end users
RQ3-D5	Easy-to-install and execute	Aims to discover if VPEs can be installed and executed without the support of high-tech scribes
RQ3-D6	Liveness and simulation	Determines the level of responsiveness of these VPEs to the programmer edits as well the available simulation capabilities
RQ3-D7	Evaluation methods	Aims to identify which tools have been evaluated with real end users and which techniques were used for these evaluations
RQ3-D8	Participation of end users	Aims at defining the degree of participation that these tools enable for their target end users, such as design time, use time or both

to analyze the software approaches used to build VPEs for Social Robotics, from which modular and reusability capabilities can be inferred. Dimensions RQ3-D3 to RQ3-D5 aim at discovering which VPEs tools enable the support of the entire life-cycle of application development. For this purpose, the user must be able to install, configure, and use these VPEs and create their own interactive scenarios with their robots in their own computing devices without the help of high-tech scribes. RQ3-D6 aims to identify the levels of liveness supported by these VPEs as well as simulator tools supporting them. *Liveness* is a concept used in the literature for referring to the capabilities of programming environments to provide an immediate feedback cycle [81]. This feature can reduce the cognitive burden on programmers and enable users to adopt a more exploratory programming style [82]. According to [83,84], it is possible to identify 4 liveness levels: *level 1* (informative), where visual representations understandable only for expert developers are provided; *level 2* (informative and significant), where visual representations of the programs have enough information to enable their execution; *level 2* (informative, significant and responsive), where feedback can be provided on demand with a “run” button; *level 4* (informative, significant, responsive and live), where feed-

back is automatically provided as edits are done in the program. Unlike programs executed on a computer, robots can act and modify their environment. Therefore, feedback requiring the robot to perform motions needs special care. A safe option to implement level 4 of liveness is through simulations. RQ3-D7 aims at identifying the methods used to evaluate the suitability of these VPEs in this sense. Finally, RQ3-D8 aims to assess which VPEs have been reported as: (i) only tested or evaluated by their developers and/or colleges (e.g., by students pursuing engineering studies), (ii) used by real end users at design time (i.e., in laboratories); (iii) used by real end users at run time (i.e, in-the-wild conditions).

(RQ4) What are the open issues and challenges for VPEs in the domain of Social Robotics? This research question is mostly addressed in Section 8, based on the observed values of dimensions in RQ1, RQ2 and RQ3.

4.2. Search process

The search was carried out in well-established databases in the field of intelligent robotics systems, namely IEEE Xplore, Science Direct, ACM Digital Library, Springer Link and Web of Science. Examples of other systematic reviews focusing on Robotics applications and methods using these sources are [80,85,86]. The time period of publications covered is between 2008 and 2018. The year 2008 was chosen as the starting year as no earlier tools are described in [16] and [20]. Moreover, 2008 is just before two major events in Robotics, which are relevant for the focus of this article. The first is the initial release of the Robot Operating System [87] in version 1.0 (2009), which become a milestone in academic robot development. Approaches using ROS-based frameworks are nowadays quasi-standard for many researchers in Robotics. The second is the release of the first commercial version of Nao social robot (2008) and its official EUD tool [88]. Nao is probably the most successfully used social robot up to date, which is evidenced by the fact that most of the VPEs reviewed in this article support this robot.

In order to obtain key terms for the search string, we applied three different strategies: (i) an analysis of our main goal and the research questions, (ii) an analysis of core articles from previous state-of-the-art studies, and (iii) pilot testing. For step (i), our goal was to identify relevant and recent “End-User Development” or “End-User Programming” supporting “Visual Programming Languages” for “Social Robotics”. These keywords are also contained in our main research question (RQ1). Therefore, we extracted “End-User”, “Programming”, “Visual”, “Development” and “Robot”. In step (ii), we used the SEOBook keyword density analyzer [89] to identify the most recurrent words in two well-known core papers of EUD for Social Robotics, specifically [62] and [88]. Based on this analysis, such keywords such as “Robot”, “User”, “Development” and “Programming” were found to be relevant. In step (iii), we executed and refined the keywords and the search string iteratively. This process was validated using a quasi-gold standard [90]. Finally, the main keywords used for the search were “Robot”, “End-User Development” and “Visual Programming”. A correlated keyword for “End-User Development” is “End-User Programming”, and one for “Visual Programming” is “Visual Language”. The search string was defined using the Boolean operators as follows: *‘Robot’ AND (‘End-User Development’ OR ‘End-User Programming’ OR ‘Visual Programming’ OR ‘Visual Language’)*.

4.3. Selection of papers

The next step in the review protocol was a clear definition of the criteria used to decide which papers were used in this review, and how and when those criteria were applied. The inclusion (IC) and exclusion (EC) criteria for this study are shown in Tables 5 and 6, respectively.

Table 5
Definitions of inclusion criteria.

Criterion	*Description*
IC 1	The focus of the article is to describe an EUD or EUP tool for Robotics
IC 2	The presented tool is focused on supporting end users and not expert developers nor people working on industrial robots

Table 6
Definitions of exclusion criteria.

Criterion	*Description*
EC 1	The article only presents an Application Programming Interface using a purely textual programming language rather than an EUD or EUP tool implementing a VPL
EC 2	The main focus of the presented tool is another EUP approach, such as NLP, tangible programming, or PbD, and not the use of a VPL
EC 3	The presented VPE is technically limited to be used in robot toys or kits and for STEM educational proposes
EC 4	The article is not written in English

The following steps indicate how and when the defined inclusion and exclusion criteria were applied: (1) reading the title, abstract and keywords of all articles applying inclusion criteria IC1 and IC2; (2) reading the introduction, contributions, and conclusion of studies included in Step 1 to eliminate irrelevant documents which meet some of the exclusion criteria; (3) a complete reading of the remaining studies in Step 2 to validate their relevance; (4) collecting all the useful information for the proposed research questions. The performed search process is graphically illustrated in Fig. 1. As shown in the figure, a total of 1010 articles were returned by an automatic search in the selected databases. From these entries, 54 were selected after executing step 1. In step 2, after performing skim reading, 33 were excluded. Finally, 21 articles were selected for this review. However, some articles, such as [91,92], reference the same interface in different development steps. Therefore, in step 3 we identified more relevant and complete articles describing these VPEs. Finally, a total of 16 interfaces were selected for this review. Nonetheless, we went through all the 21 articles to perform data collection.

4.4. Limitations of the study

The validity of the review may be limited by three factors, which are described in [93,94].

Publication bias is described in [94] as the problem that “positive results are more likely to be published than negative results”. In this review, only a few of the selected papers report negative results. However, the interpretation of positive or negative results often depends on the point of view of each researcher [94]. A standard method used to deal with this issue is scanning the gray literature, i.e., M.Sc. and Ph.D. theses, books, workshop proceedings, and technical reports. However, there still exists a risk that the presented analysis in this article does not offer a complete overview of the reviewed VPEs.

Interpretive validity is achieved when the derived conclusions are reasonable given extracted data [93]. For this, three researchers experienced in areas such as Software Architectures for Robotics, Artificial Intelligence, Social Robotics, Usability Engineering were involved in the validation of conclusions.

Theoretical validity is determined by the ability of researchers to capture the intended data [93]. The search process was conducted

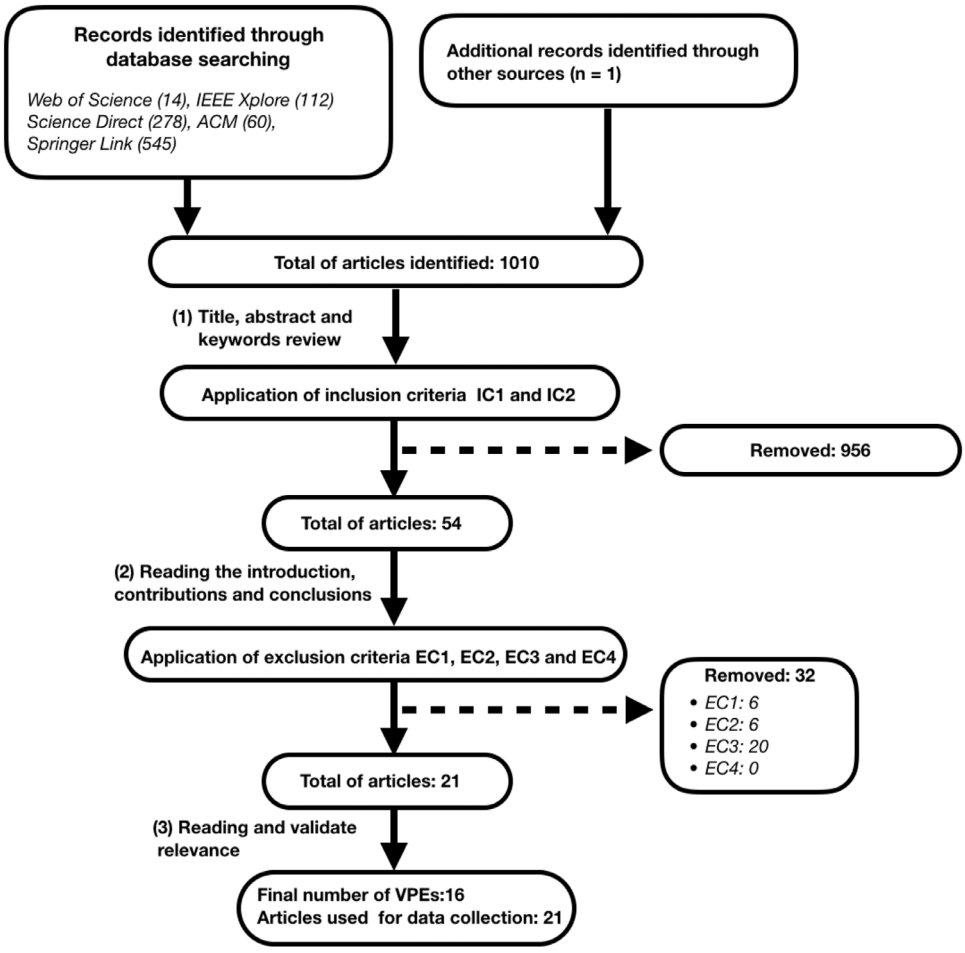

Fig. 1. Flowchart of the search strategy.

by an individual author, which is the main threat to validity. Therefore, during the inclusion and exclusion phase, there is the possibility that some VPEs might have been missed. In order to reduce this risk, we asked experts if they knew of any unpublished results or other relevant sources not initially considered in this review. During data extraction analysis and classification phases, the researcher bias is also a risk. To reduce this bias, three independent reviewers assessed all extractions made by the one reviewer, such as suggested by Petersen et al. [93] and Brereton et al. [95] and applied in [93]. However, and as described in [93], this risk cannot be eliminated completely as it involves human judgment.

4.5. Reporting of results

Next, we answer the research questions of this study based on the presented dimensions. Research questions RQ1 is answered in Section 5 by presenting a brief overview of the VPE tools for Social Robotics found after following the proposed search protocol. Section 6 is used to answer the research question RQ2 by discussing the AAI-based tools found in the resulting articles. Research question RQ3 is addressed in Section 7 by presenting and analysing the software tools used in the development of VPEs for Social Robotics. Finally, research question RQ4 is answered in Section 8, which presents the identified open challenges.

5. VPEs for Social Robotics (RQ1)

In order to answer RQ1, this Section presents a brief description of the VPEs resulting from our systematic search and analysis. We classify these VPEs in three categories, namely *dataflow-based, block-based* and *form-filling*. This classification was explained in Section 3. Table 7 shows the general features and the targets of these VPEs.

5.1. Dataflow-based interfaces

The Microsoft Robotics Developer Studio (MRDS) [96] provides a VPE oriented at enabling novice and expert programmers to generate robot-based applications in Microsoft Windows. MRDS is based on C#, includes a 3D simulator and allows for distributed messaging between different modules using a SOAP-based application layer protocol called *Decentralized Software Services Protocol* (DSSP). MRDS can be used with a set of commercially available robots, including Nao and Kondo KHR-1 humanoid robots. However, the support for MRDS has been discontinued recently.

Choregraphe [88] is a cross-platform and desktop-based VPE developed by Aldebaran Robotics (now Softbanks Robotics). Its programming approach is based on using different wires or connectors to organize multiple robot behaviors in sequence or for parallel execution. Choregraphe includes a 3D simulator and allows for the design and debugging of animations of robots using a timeline interface. Furthermore, it enables designers to develop low-level scripts based on Python 2, and also to create high-level modules (denoted as *boxes*), which can be saved as libraries for later reuse. The editing of each box parameters can be done using form-based visual interfaces.

The Tino's Visual Programming Environment (TiViPE) [62] is a desktop and data-flow interface built on QT [108]. Originally, it

Table 7
General features of VPEs for Social Robotics.

Name	*Target users*	*Application domain*	*Robots*
CodeIt! [97]	Novice and expert programmers	Service robots	Sovioke Relay, Turtlebot
OpenRoberta [98]	Children, teens	Education	Nao, toys
Robokol [99]	Therapists	Robot-based therapy	Ono
BEESM [100]	Novice and expert programmers	Smart environments	Turtlebot
RIZE [101]	UX/UI designers	Long-term and social HRI, child–robot interaction, entertainment	Many
ProCRob [102]	Teachers, therapists	Robot-based therapy, tutoring	QR
MRDS [96]	Novice and expert programmers	Autonomous vehicles, competitions, entertainment	Many
Choregraphe [88]	Novice and expert programmers	Social HRI, entertainment, robot-based therapy	Nao, Pepper, Romeo
TiViPE [62]	Therapists	Robot-based therapy	Nao
Interaction composer [92]	UX/UI designers	Shopping malls	Many
RoboStudio [103]	Novice and expert programmers	Healthcare	iRobiQ-S
RRP-VPE [104]	Novice and expert programmers	N.A.	Nao
RoVer [105]	UX/UI designers	Social HRI	Nao
Interaction blocks [106]	UX/UI designers	N.A.	Nao
English2NAO [107]	Therapists	Robot-based therapy	Nao
PersRobIoTE [67]	Novice programmers	Smart environments	Pepper

was designed to enable rapid prototyping of robot behaviors using a massively parallel processing and cross-platform approach. In TiViPE, modules can be developed using different programming languages, and can be integrated with their documentation in a stand-alone executable, each one characterized by its own graphical front-end. The development of abstract and complex modules in TiViPE (i.e., made up of simpler, basic, modules) can be done using a form-based interface to combine selected modules. Then, such modules can be reused in the same or other TiViPE-based programs. The only robot supported by TiViPE is Nao. However, unlike Choregraphe, TiViPE allows for the use of multiple Nao robots at the same time specifying their IP address. In TiViPE, the overall robot behavior's control flow is organized using *one-to-many* connections between a set of input/output ports in each graphical module, i.e., one output port of a module can be connected to more than one input ports of different modules. A set of optional ports in TiViPE-based modules can also be defined to specify relevant parameters needed for subsequent execution in other modules. The latest available version of TiViPE also allows for the development of sensory-driven dynamic and parallel behaviors, which can be defined using a domain-specific control language. The main real-world use cases in which TiViPE is employed are related to robot-based social therapy.

Interaction Composer [92] is a flowchart-like and interaction-oriented design framework specifically aimed at facilitating the development of social robot applications via coordinating cross-disciplinary teams of expert developers and UX/UI designers, i.e., end users and researchers in social sciences. As envisaged by the original proposers, it is necessary to clearly separate the role of different professional participants in the team. Expert software developers are in charge of low-level programming activities, such as data processing, interfacing with hardware equipment, and the development of basic robot behaviors in C++, whereas interaction designers only focus on defining the interaction workflow and dialogue generation. Interaction Composer is characterized by a 4-layer, modular architecture enabling the use of different robots sharing a number of similar features and capabilities. Besides the standard interaction workflow, this framework allows for specifying interruptions when certain conditions are met. When an interruption is handled, the control flow resumes from the point where the interaction workflow was interrupted. Furthermore, Interaction Composer allows encapsulating visual elements in a hierarchical way. However, notwithstanding this hierarchical approach, the authors recognize a number of issues related to scalability and flexibility in their dataflow-based interface [91]. The framework has been widely adopted for developing Social Robotics applications in real-world settings, such as shopping malls and supermarkets, and also in situations when a robot acts semi-autonomously [91]. The public availability of Interaction Composer has been discontinued since 2011.

RoboStudio [103] is a desktop-based VPE aimed at the design and development of service robots for medical and healthcare applications. Built on top of the Healthbots framework [109], RoboStudio has been developed using Java and Netbeans, which enable cross-platform support and a low memory footprint. It allows for using software components originally developed for ROS [87,110], and OpenRTM [109]. As a consequence, RoboStudio is characterized by a high flexibility for the integration of robots and distributed sensors. The design of the application workflow is based on concepts borrowed from Finite State Machines (FSMs). As a consequence, the main programming interface has been designed for expert developers rather than for novices. However, the interface design can be reused and extended to embed other VPEs. A simple example of this is discussed in [103], where a novel interface, called LTLCreator, has been developed on top of modules originally developed using RoboStudio and Netbeans. Algorithms developed using RoboStudio can be converted to an XML-based domain-specific language called Robot Behavior Description Language (RBDL), which can be executed using the Healthbots execution engine [111]. Unfortunately, RoboStudio is not available for use to date.

The Reactive Robot Programming – Visual Programming Environment (RRP-VPE) [104] is a dataflow- and web-based interface powered by Node.js [112]. RRP-VPE is based on the *Reactive Robot Programming* paradigm, which can be described as a "declarative programming approach towards the development of event-driven applications built around the notion of continuous time-varying of data streams and the propagation of change" [113]. Unlike most VPEs based on component-based software engineering approaches, RRP-VPE advocates an approach in which modules can be devel-

oped in the same environment rather than using a separate, independent tool. RRP-VPE is based on the notion of *RRP graphs*. These graphs define processes aimed at transforming inputs into outputs using a set of different connectors organized in a possibly complex structure. Examples of such connectors include operators to map certain data to given functions, filter inputs, sample data, or to merge different data sources according to a given logic. However, in order to use RRP-VPE, users are required to be quite skillful in low-level software development to fully understand the notation related to the declaration of variables, data assignment, and reactive programming operators. It is not clear whether non-technical end users can adopt the novel concepts used in RRP-VPEs, and what are the usability and cognitive issues implied. RRP-VPE is available online as open-source software in [114].

RoVer [105] is an authoring VPE designed with a two-fold purpose in mind: firstly, to enable prototyping of human–robot interaction scenarios built on top of a number of available interaction primitives, and, secondly, to encode appropriate social norms, possibly not known to the designers *a priori* but that emerge during the interaction. RoVer adopts formal verification techniques to ensure that the developed programs satisfy a set of social norms encoded as logical rules. To this end, RoVer employs the Prism Model Checker [115]. Moreover, this framework is able to provide designers with feedback when a certain social norm cannot be met. Analogously to other frameworks aimed at human-robot (social) interaction, RoVer adopts small behavioral primitives, called *microinteractions*, which can be aggregated to work sequentially or in parallel. Microinteractions can be aggregated in groups, which are organized as a set of states. Then, the overall human–robot interaction unfolds using a structural, FSM-based architecture, in which transitions between groups of microinteractions depend on the current robot beliefs. RoVer is implemented in Java and can work in Linux or OSX operating systems. Currently, it has been used with the Nao robot. RoVer is available online [116].

Interaction Blocks [106] is a visual authoring environment aimed at the fast prototyping of human–robot interaction processes using Nao. The application uses a set of predefined interaction patterns as basic building blocks to generate more complex interactive processes also sequenced in a time-line fashion. These patterns have been selected by observing different human-human interactions in typical, social settings, e.g., conversations, collaborations, instructions, interviews or storytelling. The main capability exhibited by Interaction Blocks is an easy integration between human–robot interaction patterns and text-to-speech, speech recognition and appropriate gaze behaviors for robots. However, this tool is not available online. The original authors of Interaction Blocks have considered the lessons learned during its development for the design of RoVer [116].

5.2. Block-based interfaces

The Programming Cognitive Robot (ProCRob) environment [102] is a full-fledged software architecture designed to support the development and customization of applications in which social robots are used by teachers and therapists. The architecture has been applied mainly to support innovative therapies for children suffering from the Autistic Spectrum Disorder (ASD) by means of a ROS-based humanoid robot called QT. The ProCRob's architecture is composed of three layers: the first is a functional layer implemented in ROS or YARP [117] made up of software components enabling such basic social skills as gesture expression, text-to-speech, as well as speech, face and object recognition; the second is a middleware embedding a domain-specific language called Robot Agent Programming Language (RobAPL), which uses a Prolog-style rule- and logic-based approach to define goal-oriented behaviors using high-level abstractions and the *Belief-Desire-Intention* (BDI) model [118]; the third is a front-end VPE based on Google Blockly. ProCRob allows its users to represent and manage robot plans based on a set of tasks organized sequentially or in parallel on the basis of *a priori* commands or external events. The basic workflow unit is called *play*, which is represented by a behavioral block embedding text, audio, face expressions, and body animations. Unfortunately, ProCRob is not available online.

CustomPrograms/Codeit! [97] is a Google Blockly and web-based interface designed to reproduce the expressiveness of general-purpose programming languages by the use of low-level constructs such as loops, variables, math utilities and functions. Built on top of Node.js and roslibjs [119], it also provides a set of high-level programming abstractions denoted as *primitives*. However, it is explicitly mentioned in [120] that the use of general-purpose programming language constructs, while they can be easily and intuitively used by experienced programmers, require more training and generate more complex systems when used by inexperienced programmers. CustomPrograms/CodeIt! has been used for a series of end-user applications with mobile service robots in exhibitions, hotels [120], and for STEM-based training programs [121]. It is noteworthy that one of the main advantages of this interface is its compatibility with ROS-based software modules. However, one of its significant drawbacks is the impossibility of reusing code due to the limitations of the default features of the Google Blockly library. A study evaluating the ease-of-use and expressiveness of CustomPrograms/CodeIt! is reported in [120]. This framework is available open source [97].

OpenRoberta [98] is a block-based VPE mainly used for educational aims. However, this VPE has also been used in end-user applications [122]. OpenRoberta is based on Google Blockly, and enables software development for a variety of toy robots, single-board micro-controllers, and the social robot Nao. Unlike most VPEs analyzed in this article, OpenRoberta comes in two versions. In the first version, it can be run as a browser app connected to the Internet using a cloud-based server as a back-end. This option simplifies to a great extent installation and setup. In the second version, it is based on an offline, Java-based and cross-platform local server. However, due to its mainly educational-oriented target, OpenRoberta exploits many low-level development abstraction primitives, which must be grounded to the use of classical programming abstractions. This approach is in fact more suitable for educational purposes.

Robokol [99] is oriented to non-programmers, and is focused particularly on the development of applications in support for ASD-related therapy. The Robokol's interface is powered by Snap! [76,123], and enables cross-platform support. Such support is possible by connecting external devices to a data exchange server (e.g., a remote computer running ROS). This connection can be established by a plug-and-play approach (i.e., the device advertises its own description rather than requiring a user-specific setup) via websockets using the ROSbridge protocol and suite [119]. Experimental settings where Robokol has been adopted are related to the use of the Ono social robot, as well as a therapeutic device called the *Sensory Sleeve* [99]. Like CodeIt!, Robokol uses general-purpose programming language abstractions. The framework does not seem to be available online.

The Block-based End-user programming tool for SMart Environments (BEESM) [100] is a VPE framework based on Google Blockly. The application allows for rapid prototyping of applications involving smart environments, microcontrollers and mobile robots. Like Robokol and CodeIt!, the back-end is based on ROS, and the whole framework mainly adopts low-level general-purpose programming notations. This low-level abstraction enables the users to learn PHP and how to code with Arduino boards, which is required to program smart environments and mobile robots with the supported middlewares and libraries of this interface. It also includes a 2D

simulator for smart environments and mobile robots. It is reported that the BEESM interface will be evaluated in usability tests soon. However, BEESM is not available online yet.

Our own tool, the Robot Interfaces from Zero Experience (RIZE) framework [101] is a cross-platform, block-, form- and web-based interface enabling remote control and the generation of intelligent authoring behaviors for different robots. RIZE is built on top of the NodE Primitive (NEP) programming framework [64], which abstracts the transport layer to support distributed and modular systems using different middlewares, message libraries (e.g., ROS, ZeroMQ [124] and nanomsg [125]), and communication patterns. Unlike the majority of block-based interfaces based on Google Blockly, RIZE does not adopt general-purpose software development abstractions. On the contrary, it uses a modular approach based on the definition of independent behaviors that can be easily reused in other RIZE-based programs. Robot behaviors are encoded as *behavior trees*, i.e., a meta-architecture for the generation of reactive, modular, and complex agents [38], and are executed by a decision-making engine. RIZE has been used for the remote control and the generation of intelligent behaviors using a ROS-based Turtlebot Burger robot, Nao and Pepper humanoid robots, as well as a robot manipulator built with Dynamixel servomotors and controlled in Matlab/Simulink. Real-world applications include museum exhibitions and theater performance [64], child–robot interaction [126,127], long-term human–robot interaction experiments in home settings and research in emotional intelligence for robots [128]. RIZE is available online [101].

5.3. Form-filling interfaces

English2NAO [107] is an EUP tool in which programming inputs can be set both by natural language processing and with a form-filling interface for enabling the therapists to create programs for NAO robots. This interface is developed as a web-based application using Django [129], and runs on top of the TiViPE engine. This EUP tool was developed to overcome some of the usability problems presented by TiViPE [107]. Online availability of this English2NAO cannot be assessed.

PersRobIoTE [67] is a web-based, form-filling interface. It adapts the Trigger-Action Programming (often used in EUD tools for IoT scenarios) paradigm for allowing the creation of applications involving Pepper robots. The users of PersRobIoTE need to define a set of rules, which are mainly composed of triggers (i.e., conditions concatenated by and/or Boolean operators) and actions. These rules are encoded in the JavaScript Object Notation (JSON) formalism, and are created and managed by a decision-maker module called the Rule Manager. Moreover, it uses backboard-like modules, referred to as their authors, such as a Context Manager to handle perceptual inputs from both Pepper robots and IoT devices. The communication between these modules is carried out using the Server Sent Events (SSE) framework [130]. However, PersRobIoTE is not available online yet.

6. Modeling intelligent behaviors for social robots (RQ2)

In this Section, we address research question RQ2, which aims at discovering and analyzing the AAI-based tools used for supporting EUP and EUD in Social Robotics. RQ2 is addressed in three ways: (i) presenting a general description as well as advantages and drawbacks of those AAI-based tools for supporting VPEs, (ii) analyzing the modular capabilities of these AAI-based tools, and (iii) identifying the abstraction levels (i.e., the programming primitives) generally used in these approaches. The values of dimensions used in RQ2 are shown in Table 8. The table also includes the dimension RQ1-D2 (EUP approach) for comparative purposes.

6.1. Scripting-based

As shown in Table 8, most block-based programming VPEs, such as Open-Roberta, Robokol, and CodeIt! use general purpose scripting to enable the creation of applications with social robots. In this approach, end users need to become familiar with classical scripting approaches such as *if-then-else* conditional statements, *for* loops, creating variables and functions, and using low-level mathematical and logical operations. The acquisition of these low-level programming skills is a major objective of STEM educational courses. However, the suitability of this approach for enabling the creation of intelligent robots by end users can be hindered by usability and code reusability issues. As described in [50], the Cognitive Dimension framework suggests that using too many low-level programming notations may produce usability problems associated with high viscosity (i.e., users need to manipulate many elements to accomplish a task), and may require hard mental operations and distant mapping to the problem domain of social interaction. Moreover, using general descriptions for programming agents may produce code that is hard to reuse and maintain [38] (Fig. 2).

6.2. Rule-based

The first computer games and robot-based systems used modeling and programming frameworks based on rules for building intelligent behaviors. This approach is simple to implement and presents a uniform representation method, which can be intuitively used by non-programmers [131]. However, these systems also present many relevant drawbacks. While non-programmers can easily grasp the approach of using individual rules for programming robots, they may face difficulties in going beyond the declarative approach based on rules. They also may have difficulty in understanding the implications of multiple rules, some of which may conflict. Moreover, the lack of structure in rule-based systems often leads to (i) maintainability issues and error-prone handling of programming elements (rules) in complex systems [40], and (ii) unstable and unexpected behaviors when creating very large and complex programs [132]. While programming approaches using disjoint and priority-based rules are currently very popular for developing IoT applications, they have been widely replaced by more structured and robust AI-based architectures, such as Finite State Machines or Behavior Trees, in areas requiring the development of more complex, social and interactive agents. Many VPEs using rules are developed using form-filling programming environments. Such an approach is especially popular in IoT applications. In this context, the approach used on PersRoboIoTE is inspired by EUD solutions for IoT systems, such as IFTTT [133]. Fig. 3 shows a simple visual notation for systems using rule-based systems and form-filling. For these VPEs, the main programming task of end users is to select a condition or a set of conditions (concatenated with logical AND/OR operators) that will trigger appropriate robot action.

6.3. State-based

By including the notations of states and transitions (i.e., a decision logic that makes a system to change from one state to another), as well as adding a structure to a set of disjointed rules, a rule-based system turns into a state-based method. The most popular approach used to model state-driven systems is constituted by Finite State Machines (FSMs). FSMs are represented as directed graphs, where each node of the graph represents a state. In a FSM, each transition to a new state represents an event. These events can trigger the execution of some specific script or sequence of robot actions. In general, FSMs are robust and easy to understand

Table 8
Comparison between AAI approaches using in VPEs (RQ2).

Name	*EUP approach*	*AAI approach*	*Programming primitives*
CodeIt!	Block-based	Scripting	Hardware, Algorithm, Social
OpenRoberta	Block-based	Scripting	Hardware, Algorithm, Social
Robokol	Block-based	Scripting	Hardware, Algorithm, Social
BEESM	Block-based	Scripting	N.A.
RIZE	Block-based	Behavior-based	Social
ProCRob	Block-based	Behavior-based	Social, Emergent
MRD	Data-flow	N.A.	Hardware, Algorithm
Choregraphe	Data-flow	State-based	Hardware, Algorithm, Social
TiViPE	Data-flow	State-based	Hardware, Algorithm, Social
Interaction Composer	Data-flow	State-based	Hardware, Algorithm, Emergent
RoboStudio	Data-flow	State-based	N.A.
RRP-VPE	Data-flow	Behavior-based	Hardware, Algorithm
RoVer	Data-flow	State- and rule-based	Social
Interaction Blocks	Data-flow	State-based	Emergent
English2NAO	Form-filling	State-based	N.A.
PersRobIoTE	Form-filling	Rule-based	Social

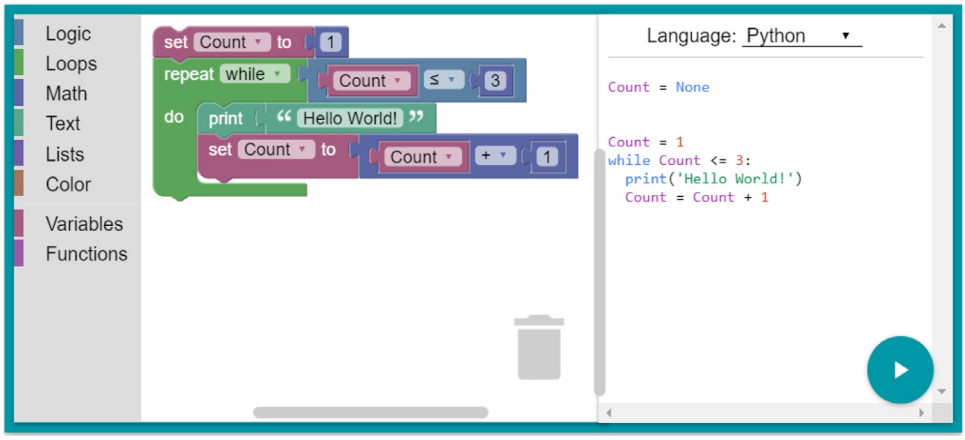

Fig. 2. Example of a block-based programming environment using general purpose programming notations and Google Blockly. In this approach end-user code is converted to real code of some programming language.

Fig. 3. The simplest notation used in a rule-based VPEs.

even for novice end users [134]. However, the use of FSMs inherently implies some reactivity and modularity issues, which are analogous to those associated with *goto* statement [40]. Both the *goto* statement and FSMs can be considered as a “one-way” control transfer (i.e., the control flow jumps to another section of the program), which is described in [40,135] as “too much an invitation to make a mess of one’s program”. As described in [40], this leads to a trade-off between reactivity and modularity of a programming system. In order to create a reactive and complex social interactive application, a program built using FSMs requires too many one-way control transitions between visual elements. As shown in Fig. 4, this results in very tangled diagrams where the modification or removal of some elements may need checking every transition and state associated with that component.

6.4. Behavior-based

By hierarchically organizing and separating the decision logic from the behavior code, a state-based system turns into a behavior-based approach. Fig. 5 shows the main difference between state-based and behavior-based modeling methods.

As shown in [40,70] most behavior-based modeling methods used in Robotics can be generalized by a Behavior Tree (BT). Unlike FSM, BTs are considered a “two-way” control transfer, i.e., after the execution of an event or function, the control flow returns to the calling part of the program, which enhances modularity [40]. A typical BT implementation is composed of two types of nodes, namely operators and terminal nodes. Fig. 6 shows an example of a simple BT. While operator nodes (in white) are used to perform control flow and behavior selection, terminal nodes (in blue and gray) define and check preconditions and execute the proper behaviors. The most basic operators in BTs are *Sequence* and *Selector*. The functionality of these and other common operators in BTs are described in [40]. The execution of a BT follows a classical “depth-first” traversal order from the root node to some terminal node. After the activation of a node (when the BT traversal algorithm reaches the node), this node is assigned a status, which can be “success”, “failure” or “running” depending on the node type. Each iteration of the BT traversal algorithm performs decision-making tasks depending on the status of these nodes. By definition, BTs are also modular and reusable [70], as each branch of a BT can be considered as an independent module. Fig. 6 shows four possible modules that can be easily reused in other programs. Unlike FSMs, BTs have just started to gain attention in Robotics. Therefore, available software frameworks supporting this AAI-based approach are less mature [40,64]. Moreover, concepts involved in the creation of BT can be difficult to understand by end users, as it is required to learn the “depth-first” traversal graph search and many low-level operators for performing decision-making. An alternative aimed at

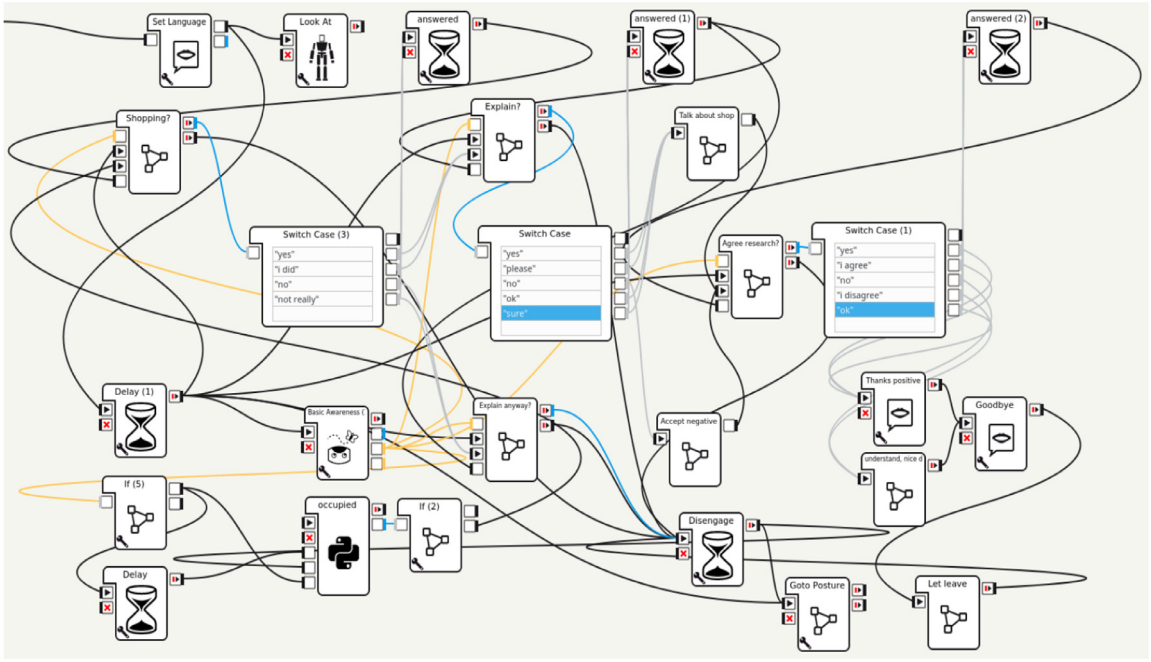

Fig. 4. Example of spaghetti code in a dataflow-based VPE (example taken from [136].

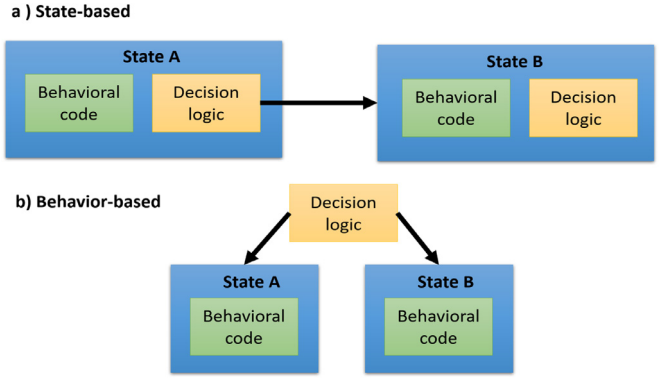

Fig. 5. In state-based methods (a), each state requires the definition of the decision logic that indicates the decision-making system how to change to another specific state. Behavior-based approaches (b) separate decision logic from behavior code enabling a hierarchical and modular representation (adapted from [132]).

enabling the use of BTs for end users has been proposed in RIZE by changing the way BTs are modeled. In this approach, rather than allowing an end user to build and execute BTs using a tree structure and low-level operators, the end user can build their programs by concatenating a set of BT modules or sub-BTs in a declarative way using a Google Blocky environment. These high-level modules or social primitives are built by expert programmers using a low-level, domain-specific language. More details are reported in [64].

7. Tools, technologies and evaluation methods for VPEs

As shown in Table 9, Web technologies, such as HTML and Node.js (based on Javascript), are preferred to build VPEs for Social Robotics applications. However, only OpenRoberta is built as a Web service. Instead, other VPEs using Web technologies are designed to be executed on a desktop using such server-side frameworks as Node.js and Django. In order to build block-based programming environments, the preferred tool is Google Blockly [78], which provides more features and flexibility than similar tools such as Snap [76] and Scratch [75]. As was described in Sections 3 and 6, some block-based VPEs using Google Blockly (specifically, RIZE and ProCBob) use this tool as a domain-specific language, whereby the code is executed in a more advanced AI-based architecture (BTs in the case of RIZE and Belief-Desire-Intention in the case of ProCBob), rather than in a general purpose programming tool. Moreover, older VPEs, such as Choregraphe, TiViPe, and MRD were built as *desktop-based* tools for developing user interfaces such as Visual Studio and Qt. The only recent VPE reported to be designed as a classical *desktop-based* interface is RoVer, which was implemented in Java.

Table 9 shows that many recent VPEs use some CBSE frameworks, with ROS being the most popular. In this approach, software modules are seen as isolated processes or nodes that are executed in parallel and that can be developed in different programming languages. This approach enables an easy reuse of many open-source software tools developed by the Robotics community, thereby creating more robust and complex robot systems. However, most of these recent VPEs, such as CodeIt!, Robokol and BEESM, require the execution of a server module or a node in a computer with the right version of Ubuntu installed. This can be a barrier to their adoption by end users for use-time design activities. This is mainly due to a steep learning curve associated with ROS [137]. Drawbacks of ROS for EUD were described by the creators of TiViPE in [138] as: (i) most of end users are Windows users and require easy-to-install software tools; and (ii) they hardly understand (without training) many of the concepts required to use ROS. Therefore, a high-tech scribe skilled in ROS is often required for the installation and execution of the web server and for maintaining or extending these VPEs, which can be performed by launching additional ROS nodes. These issues are still not solved with ROS 2.0, as it requires following complex steps for its installation in Windows 10 and training for their use. Due to these issues, interfaces such as TiViPE, MRD, and Choregraphe have been developed as monolithic applications, which are easy to install for the average end user, and where software modules are executed in different threads. This can explain why only VPEs characterized by easy-to-use wizard installers have been reported as enabling both the cre-

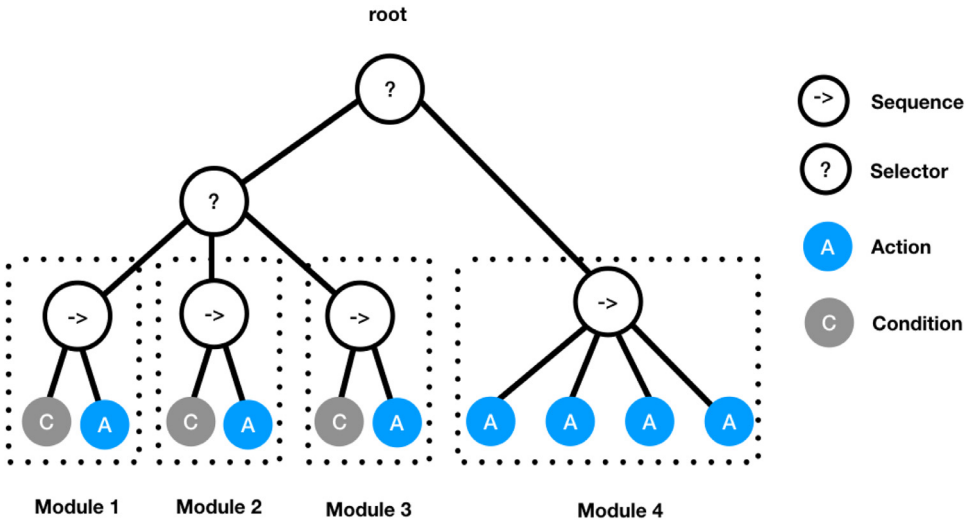

Fig. 6. Example of a behavior tree.

Table 9
Dimensions used for answer RQ3.

Name	*Communication*	*Software*	*Accessibility*	Operating Systems	*Easy-to-install and execute*	*Reported liveness*	*Evaluation methods*	*Participation of end users*
CodeIt! (2017)	ROS, rosbridge	Blockly, Node.js, HTML	Online	Server in Linux	Requires support of high-tech scribes	Level 3	Quantitative	engineering students and end users at design-time
OpenRoberta (2014)	POSIX socket	Blockly, HTML	Online	Internet-dependent	No installation required	Level 2 / 2D simulator	none	children at use-time
Robokol (2016)	ROS, rosbridge	Snap, HTML	N.A.	Server in Linux	N.A.	Level 3	none	N.A.
BEESM (2018)	ROS, rosbridge	Blockly, HTML	N.A.	Server in Linux	N.A.	Level 2 / 2D simulator	none	N.A.
RIZE (2019)	NEP	Blockly, Node.js, HTML, Vue.js	Online	Windows, OSX, Linux	End-user wizard-like installers	Level 3	none	comedians, interaction designers at use-time
ProCRob (2017)	ROS, YARP	Blockly, HTML	N.A.	Server in Linux	N.A.	Level 2	none	end users at use time
MRD (2007)	POSIX socket	Visual Studio	Discontinued	Windows	End-user wizard-like installers	Level 3	none	N.A.
Choregraphe (2009)	POSIX socket	Python	Online	Windows, OSX, Linux	End-user wizard-like installers	Level 4 / 3D simulator	none	interaction designers at use-time
TiViPE (2011)	POSIX socket	Qt	Online	Windows, Linux	End-user wizard-like installers	N.A.	Cylomatic complexity, Cognitive Dimension	interaction designers at use-time
Interaction Composer (2012)	POSIX socket	N.A.	N.A.	N.A.	N.A.	N.A.	none	N.A.
RoboStudio (2017)	ROS, ROCOS, OpenRTM	N.A.	N.A	N.A.	N.A.	N.A.	none	N.A.
RRP-VPE (2017)	N.A.	Node.js, HTML	Online	N.A.	Requires support of high-tech scribes	Level 3	NASA-TLX	engineering students at design-time
RoVer (2018)	N.A.	Java and Prism Model Checker	Online	OSX and Linux	Requires support of high-tech scribes	Level 2	SUS	engineering students at design-time
Interaction Blocks (2014)	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.	SUS	interaction designers and engineering students at design-time
English2NAO (2018)	N.A.	Django, HTML, SQLite	N.A.	N.A.	N.A.	N.A.	Cylomatic complexity, Cognitive DimensioN	therapists at design-time
PersRobIoTE (2019)	Server Sent Events	HTML, IoT	N.A.	N.A.	N.A.	Level 2	SUS	end user at design-time

ation of applications at design time and their redesign at use-time by real end users. In order to communicate with external modules, VPEs built as monolithic applications generally use simple POSIX sockets for limited tasks.

From Table 9, it is possible to see that the most popular method used to validate the suitability of these VPEs is the System Usability Scale (SUS) [139], which allows a reliable and valid evaluation of usability for a wide variety of products and services such as hardware, software, websites, and applications. Another method used to detect usability problems is the Cognitive Dimension framework, which was introduced in Section 2. The NASA Task Load Index (NASA-TLX) [140] and the Cyclomatic complexity metric [141] have also been used to measure perceived workload and complexity when creating programs for social robots. However, most of these evaluation methods are used to obtain subjective data. In the case of Codeit!, objective and quantitative evaluations, such as the time and number of successful tasks, have been performed.

8. Open challenges

This Section discusses and summarizes the current issues and open challenges characterizing VPE-based authoring tools for social robots. These are related to accessibility to external devices and resources, modularity of the human–robot interaction primitives, scalability when large programs are needed, levels of abstraction, benchmarking, explainability and control of the resulting robot behaviors, support for distributed robot frameworks, as well as simulation and debugging.

8.1. Accessibility to external devices and resources

This open challenge is based on the data obtained from dimensions RQ3-D3 (Accessibility), RQ3-D4 (Operating Systems), and RQ3-D5 (Easy-to-install and execute). An analysis of these dimensions reveals a number of accessibility issues in recent VPEs. From the point of view of non-technical end-users, such as UX/UI designers, an accessible VPE requires (i) accessibility for their use or evaluation (RQ3-D3), (ii) compatibility with end-user devices (RQ3-D4), and (iii) user-friendly installation and configuration (RQ3-D5).

Accessibility for their use or evaluation. As shown by the values obtained in dimension RQ3-D3 (Accessibility) in Table 9, more than half of the presented VPEs are not available online, even though many of them are built with open source tools. This hinders their proper evaluation and denies opportunities to obtain feedback from both the Robotic community and end users.

Compatibility with end-user devices. In many professional environments and in Academia, Linux-based systems have a good reputation and impact. However, end users who are not technically trained often assume the availability of software designed and developed for Microsoft Windows or OSX, which has a bigger market share compared to Linux-based systems for the general consumer market. While this issue is well-known and deeply understood by most commercial VPEs, such as Choregraphe and TiViPE, it is mainly ignored by most community-oriented or open-source VPEs, therefore jeopardizing their ability to gain widespread use. A common argument used to justify cross-platform support is to design the VPE's architecture as a (possibly web-based) front-end running on Microsoft Windows or OSX coupled with a back-end typically running on a Linux-based system, which is done by Robokol and the offline version of OpenRoberta. However, such architecture still requires the server-side to be configured on a Linux-based platform.

User-friendly installation and configuration. The first impression an end user has about any software tool is based on the installation and configuration phases. Therefore, these phases must be as easy and simple to complete as possible. On the one hand, such commercial VPEs as MRDS, Choregraphe, and TiViPE can be installed via user-friendly wizards. However, they are characterized by a huge memory footprint. On the other hand, most community-oriented, open source interfaces require the expertise of professional software developers for installation and configuration, which is mainly due to the necessity to setup a Linux-based system to run the server, install the third-party software from the command line, and build the required binaries. An option that enables cross-platform support and reduces the installation and configuration efforts, as well as the required memory footprint, is to run the VPE in a cloud-based server, as it is done in such educational-oriented interfaces as OpenRoberta. However, such a possibility requires a stable, permanent connection to the Internet as it depends on the online availability of the server itself. Values in dimensions RQ3-D4 and RQ3-D5 in Table 9 reveal the poor attention that open source projects have received in the creation of native and cross-platform applications that can be launched and used by end users even if they do not have access to the Internet or a Linux server. Solving these issues is relevant for enabling end users to bring robots outside the laboratory, where it is often hard to have a stable Internet connection, and where the support of high-tech scribes is not always possible.

8.2. Modularity of human–robot interaction primitives

In Computer Science, and as far as software architectures for robots are concerned, the word "modularity" is ambiguous and can be related to concepts present at different levels of the architecture and granularity scales. For this article, we consider two different meanings associated with the notion of modularity, namely *operational* and *structural* modularity. The formulation of this open challenge is based on the data obtained from dimension RQ3-D1 (Communication) in Table 9, which is related to *operational* modularity, and the analysis presented in Section 6 about modular and reusable capabilities of each AAI-based method used for modeling social robot behaviors, which is related to *structural* modularity.

Operational modularity. While a few authors consider modularity in VPEs as a simple encapsulation of function calls or a set of related functions, others aim at integrating higher-level modular abstractions, such as the *Separation of Concerns* design principle in independent processes (also denoted as nodes) and/or software packages, as it is done for instance in ROS [87]. Recently, the latter approach has been the most successful, being considered as the best practice for Robotics-related software development [142], and one that provides an increased quality in software applications. According to this approach, and as far as VPEs for Robotics-related applications are concerned, data exchange between processes is managed on the basis of a number of well-defined inter-process communication patterns [143].

Based on these concepts, and the values of dimension RQ3-D1 in Table 9, it is possible to classify VPE interfaces as having low, tight, or high operational modularity. In the first case, modules are just considered as a set of function calls. In this approach, most of the robot's sensory, perceptual, decision-making and control tasks are carried out as parts of a single process. VPEs that exhibit low operational modularity include Interaction Blocks and RoVer. In the second case, the overall robot functionality is split among different modules, and various modules communicate with each other using a *Request-Process-Reply* (or *Client/Server*) design pattern [144] through POSIX sockets. VPEs adopting this operational modularity type are MRDS, Choregraphe, TiViPE, Interaction Composer, and OpenRoberta. In the third case (also referred to as *loose coupling*), the principles of reusability, extensibility, maintainability, and robustness are enforced by the use of non-blocking and asynchronous communication design patterns, such as *Publish/Subscribe*

and *Observer*. VPEs in this class are RRP-VPE, RobotStudio, ProCRob, CustomPrograms/CodeIt!, Robokol, BEESM, RIZE.

Structural modularity. One of the main drawbacks associated with most of the analyzed VPEs, particularly those based on the flowchart concept of FSMs, is the lack of modularity, intended as a clear subdivision of roles within the resulting architecture. Such a lack of modularity-enforced design is due to the mainstream approach to organize internal workflow (i.e., mostly corresponding to the overall robot decision-making capabilities) as a single, one-way, data transfer from inputs to outputs. In a sense, such approaches replicate the somewhat classical Sense-Plan-Act architecture for robot perception, planning and control [145]. The result is characterized by issues similar to what happens with the use of the much critiqued *goto* statement, which is considered unstructured and a cause of unreliable behavior [40,146]. As a consequence, dataflow-based VPEs tend to generate *spaghetti* code (Fig. 4) and visual programs that are difficult to understand, maintain, reuse, and scale [91]. As described in [147] and reported in [91], VPEs using dataflow present three key issues: (i) their programs tend to be very large requiring the creation of too many nodes even for trivial and low-level tasks; (ii) each node requires too many inputs and links between them producing highly tangled programs (referred to as *spaghetti* code), such as shown in the example in Fig. 4; (iii) confusing iteration: the program is difficult to follow or even understand. To deal with these drawbacks, a viable solution could be to develop independent and modular systems based on two-way data exchange, e.g., hierarchical decision-making engines like BTs [148].

8.3. Scalability in large applications

In most cases, designers and developers of VPE-based authoring tools advertise use cases whereby their frameworks are adopted to design simple human-robot social interaction patterns requiring the use of few primitive behavioral blocks and connections. Like many virtual agents [39], the development of more complex or long-term applications to be delivered in everyday scenarios, such as those described in [9,149,150], can require the integration of a large number of behaviors, as well as a possibly intricate logic to coordinate their orchestration. This requirement implies a rapid increase in the difficulty in following the application's control flow, and in searching for appropriate primitive robot behaviors. Module or component encapsulation is a widely-adopted approach used in dataflow-based VPEs to deal with these issues. In many cases, however, encapsulation reduces the *mess* of dataflow-based workflow only to a limited extent [40]. Suitable design patterns to deal with these issues are rare in block-based interfaces.

8.4. Correct abstraction levels and programming notations

As analyzed in [50], and according to the values obtained in dimension RQ2-D2 (Programming primitives) in Table 8, many of the VPE-based frameworks discussed in this article are characterized by unbalanced abstraction levels in selecting robot behavioral primitives and programming notations. In fact, typical issues are related to the presence of primitives with low-level and varying abstraction levels, and to the consequence of the overall VPE usability [47].

As far as the abstraction level of VPEs is concerned, such VPEs as Choregraphe, OpenRoberta, TiViPE and Interaction Composer are characterized by several issues in usability, and in the cognitive dimension, due to the fact that they incorporate various low-level programming abstractions, which are denoted in [50] as hardware and algorithm primitives. On the one hand, VPEs including hardware primitives use graphical elements, which enable users to obtain raw data (e.g., position, velocity, sound and images) from sensory devices or actuators. On the other hand, VPEs including algorithm primitives require that users be able to provide the data generated by the hardware primitives as inputs of low-level perceptual and control modules (e.g., sound source localization, inverse kinematics, keyframe animation and face tracking). However, raising the level of programming abstraction too high, as it is done for example in Interaction Blocks, can reduce the flexibility of VPEs, and therefore the capability to create complex behaviors with robots. An alternative approach allowing for a good trade-off between the usability and the flexibility of VPEs and robot programming software aimed at generating social interactions is also described in [50]. The correct level of abstraction for developing social interaction behaviors with robots requires the use of reusable and atomic domain-specific social primitives (e.g., related to speaking, gestures, gaze, facial expressions and animations).

8.5. Benchmarking

The evaluation of interfaces with real end users is a key task required not only to show the applicability of VPEs, but also to obtain valuable data to validate or improve usability. These evaluations require data collection from both objective and subjective approaches. The collection of objective data is based on facts rather than opinions or interpretations, e.g., how many times the user makes an error, the number of times that a user has asked for help, and the task completion time. This type of data is generally collected and analyzed by those VPEs reporting usability evaluations. From the values obtained in dimension RQ3-D7 (Evaluation methods) in Table 9, it is possible to observe that the authors of TiViPE and CodeIt!/CustomPrograms have used, to different degrees, the Cognitive Dimension framework as the main tool to perform subjective data analysis. This framework is always used to identify usability trade-offs in the early stages of designs and make decisions about those trade-offs for posterior iterations [151]. While the Cognitive Dimension framework has emerged as the predominant framework for analyzing VPL, some researchers have identified some of its serious theoretical and practical limitations for its use in the evaluation and design of visual notations [152]. Some of the main issues of the Cognitive Dimension framework described in [152] are: (i) confusion or misinterpretation when interpreting and applying dimensions; (ii) lack of evaluation procedures or metrics; (iii) the omission of issues around whether the visual notations chosen are "good" or "bad" ones. A complementary approach for addressing these issues is to follow guidelines and the principles of the Physics of Notation [152], which are valuable tools for evaluating and designing visual notations. However, we observe that these guidelines and principles are often omitted in most of the reviewed VPEs. A well-accepted subjective data collection approach in Human-Computer Interaction (HCI) is the use of standard questionnaires [153] such as the System Usability Scale (SUS) [154] and the NASA Task Load Index (TLX) [155]. From the VPEs we reviewed in this article, only Interaction Blocks (using SUS), RRP-VPE (using TLX), TiVIPE/English2NAO (using Cyclomatic complexity, SUS and Cognitive Dimensions) and RoVer (using both SUS and TLX) have performed subjective data collection using standard questionnaires. Even when many of the reviewed VPEs have been used by real end users, only the designers of CodeIt!, TiViPE/English2NAO and Interaction Blocks have reported usability evaluations using real novice end users, rather than expert programmers, laboratory members or engineering students.

A comparative evaluation among interfaces, using objective and subjective data, is a valuable task in modern HCI research. This evaluation enhances the analysis and validates the suitability of proposed VPEs. However, this task is often omitted by the designers and developers of most VPEs presented in this work. Exceptions using the NAO robot are presented in [104] and [107]. Issues

that limit performing such comparative evaluations are related to the fact that (i) some of the presented VPEs are not available online, and (ii) they support different robots and target user groups. Recently, a shift of emphasis in many areas of HCI to user experience has become a central focus for interface design and evaluation [156]. However, most UX-related aspects [157], except usability, have generally been omitted in the reviewed VPEs.

8.6. Explainability and generation of robot social behaviors

Depending on the specific use case involved in the target end-user applications, robot behaviors (which can be formed by a simple action or a set of parallel robot actions requiring synchronization) used for social purposes in the papers analyzed for this survey can be classified into the following classes:

- C_1 repetitive robot behaviors that do not require any specific form of high-level intelligence nor cognitive capabilities;
- C_2 scripted sequences of basic robot behaviors, the execution of which always follows a predefined list of actions and occasionally requires user-provided input, e.g., using keyboards, joysticks, touch or speech, to continue execution;
- C_3 state- or event-based *dynamic* behaviors that do not allow for any interruption nor preemption until the current behavior is completed;
- C_4 state- or event-based reactive and dynamic behaviors enabling interruption, state change or preemption on the basis of predefined priorities;
- C_5 hybrid reactive/deliberative, intelligent behaviors that require the robot to correctly interpret and react to external and internal stimuli, and to make decisions about which actions to perform next in view of a certain goal.

While most of the VPEs reviewed in this paper can generate behaviors belonging to classes C_1, C_2, and C_3, very few are capable of generating robot behaviors which can be classified as C_4 or C_5. An exception that can be considered as a basic approach towards C_4 is Interaction Composer, which exhibits a dataflow interruption mechanism, where an interrupt could monitor some perceptual input and trigger another behavior sequence [92]. Another exception towards C_4 is TiViPE, the latest release of which includes a textual robot language that can be used to set the execution priorities of a set of serial and parallel actions. Other advanced approaches towards C_4 have been proposed by RRP-VPE and RIZE, using Reactive Programming and BTs, respectively. VPEs enabling non-programming skilled end users to design hybrid reactive/deliberative intelligent behaviors are still rare. However, it is noteworthy that while moving from C_1 to C_5 in the behavior classification, the resulting robot actions may be considered progressively less understandable and explainable for humans. This is because the composition of many simple behaviors in an intertwined chain of planned actions and reactions to certain events can lead to widely different outcomes, even when there are only small differences in the (sequences of) inputs [158,159]. It is not surprising that, as far as human–robot interaction is concerned, social robots exhibiting predictable behaviors are to be preferred given the state-of-the-art knowledge in Robotics.

8.7. Simulation and debugging

The benefits of simulation and debugging capabilities in any software development toolkit are quite obvious and do not need to be emphasized. However, human–robot interaction and the use of social robots in-the-wild are characterized by specific requirements as far as simulation and debugging are concerned. These are mostly related to the dynamic and often unpredictable nature of human–robot interaction processes and social relationships: on the one hand, it is necessary to always ensure predictable robot behavior, as well as to guarantee that the overall architecture workflow does not enter into unsafe states; on the other hand, for the robot behavior to be engaging at the social level and to ground high-quality human–robot interaction experiences, it is of the utmost importance to carry out time-consuming robot-based tests and evaluation before the final application is deployed.

In many cases these goals are not possible or easy to attain, because of practical reasons related to robot unavailability or incomplete technical development. Therefore, being able to access and leverage a high-quality, accurate, and faithful simulation of robot behavior is crucial.

Cross-platform, easy-to-use and easy-to-setup simulators are the key to increase the overall usability of VPE-based development. Values obtained from dimension RQ3-D6 (Liveness and Simulation) in Table 9, show that most of the VPEs discussed in this article lack any robot behavior and human–robot interaction simulation capabilities, the only exceptions being Choregraphe, which provides a 3D simulation for the robots commercialized by Softbank Robotics, BEESM, which includes a 2D simulator for smart environments, and OpenRoberta, which provides 2D web simulators of toy robots. Values obtained from dimension RQ3-D6 also indicate that less than half VPEs provide on-demand feedback or debugging (liveness 3), and only Choregraphe provides live feedback capabilities (liveness 4). The concept of liveness was discussed in Section 4.1 in the definition of research question RQ3.

9. Conclusions

In this paper, we presented a survey of different VPE-based frameworks to enable a EUD-based development of social robots and human–robot interaction scenarios. A structured comparison of these frameworks has been carried out from an operational point of view, classifying them as dataflow-based, block-based and form-filling. Our findings indicate that there is a need for more accessible, adaptable, modular, extendable and flexible tools and technologies to support and enable end users to become end user developers of their systems. We note that many recent VPEs are built on top of CBSE and distributed Robotics frameworks for enabling enhanced modularity and flexibility. However, the inherent complexity of most CBSE Robotics-oriented frameworks are characterised by accessibility and usability barriers, whoch makes it difficult to create EUD tools promoting independence between end users and high-tech scribes. This is because most CBSE frameworks were originally designed for supporting academic projects, and tended to have steep learning curves for their use even for expert developers. Solving these issues is necessary to enable end users to develop and redesign their applications in-the-wild. A possible direction can be the use of more lightweight, simple CBSE frameworks that (i) are adapted to the skills and resources of end users, and (ii) can be used as a glue between different software modules developed by the Robotics community and different Robotics middlewares. Moreover, our findings point to the poor attention most authors of VPEs have given towards the performance and comparative evaluation of these tools with real end users, and the need for more user studies and objective analysis of these tools using both quantitative and qualitative data.

Finally, some efforts are recently being made to overcome limitations of classical approaches using rules, scripting, and dataflow programming, thereby providing end users with more reliable, reusable and reactive programming tools to enable the creation of more complex behaviors for social robots. Unlike other information technology areas, where end-to-end black-box AI architectures are currently trending, e.g., Deep Neural Networks, AI architectures for enabling EUD of social robots mostly focus on the use of AI tools with authoring and explainable behaviors. This situation is similar

to the one facing game developers, whereby the creation of robust, explainable and suitable behaviors, in many cases defined by UX/UI designers, is more valuable than learning capabilities.

However, the creation of more complex Robotics systems will require Social Robots to learn from its environment. A possible research direction can be the use of hybrid decision-making algorithms to (i) provide a sufficient level of explainability and behavior control of agents; (ii) provide learning mechanisms enabling robots to adapt to dynamic situations; (iii) enable easy parameterization of critical aspects for developing social robots, such as personality and social norms. We hope that this information can be valuable and informative for the development of more usable and flexible VPE-based systems enabling the creation of more intelligent social robots.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Supplementary material

Supplementary material associated with this article can be found, in the online version, at 10.1016/j.cola.2020.100970 .

CRediT authorship contribution statement

Enrique Coronado: Writing - original draft, Visualization, Methodology. **Fulvio Mastrogiovanni:** Writing - review & editing, Validation. **Bipin Indurkhya:** Writing - review & editing, Validation. **Gentiane Venture:** Writing - review & editing, Supervision.

References

[1] F. Dimeas, F. Fotiadis, D. Papageorgiou, A. Sidiropoulos, Z. Doulgeri, Towards progressive automation of repetitive tasks through physical human-robot interaction, in: Proceedings of the Human Friendly Robotics, Springer, 2019, pp. 151–163.

[2] H. Canbolat, Robots Operating in Hazardous Environments, BoD–Books on Demand, 2017.

[3] L. Pu, W. Moyle, C. Jones, M. Todorovic, The effectiveness of social robots for older adults: a systematic review and meta-analysis of randomized controlled studies, Gerontologist 59 (1) (2019) e37–e51.

[4] E. Coronado, J. Villalobos, B. Bruno, F. Mastrogiovanni, Gesture-based robot control: design challenges and evaluation with humans, in: Proceedings of the 2017 IEEE International Conference on Robotics and Automation (ICRA), 2017, pp. 2761–2767, doi:10.1109/ICRA.2017.7989321.

[5] D. Mariette, B.S. Meyerson, Top 10 Emerging Technologies 2019, Technical Report, World Economic Forum, 2019.

[6] M. Niemelä, P. Heikkilä, H. Lammi, V. Oksman, A social robot in a shopping mall: studies on acceptance and stakeholder expectations, in: Proceedings of the Social Robots: Technological, Societal and Ethical Aspects of Human-Robot Interaction, Springer, 2019, pp. 119–144.

[7] J. Lindblom, R. Andreasson, Current challenges for UX evaluation of human-robot interaction, in: Proceedings of the 2016 Advances in Ergonomics of Manufacturing: Managing the Enterprise of the Future, Springer, 2016, pp. 267–277.

[8] G. Fischer, End user development and meta-design: foundations for cultures of participation, in: Proceedings of the 2012 End-User Computing, Development, and Software Engineering: New Challenges, IGI Global, 2012, pp. 202–226.

[9] M. Jung, P. Hinds, Robots in the Wild: A Time for More Robust Theories of Human-Robot Interaction, 7 (1) (2018)(2:1-2:5).

[10] M. Salem, G. Lakatos, F. Amirabdollahian, K. Dautenhahn, Towards safe and trustworthy social robots: ethical challenges and practical issues, in: Proceedings of the International conference on social robotics, Springer, 2015, pp. 584–593.

[11] F.E. Ritter, G.D. Baxter, E.F. Churchill, User-centered systems design: a brief history, in: Proceedings of the Foundations for Designing User-Centered Systems, Springer, 2014, pp. 33–54.

[12] N.B. Hansen, C. Dindler, K. Halskov, O.S. Iversen, C. Bossen, D.A. Basballe, B. Schouten, How participatory design works: mechanisms and effects, in: Proceedings of the 31st Australian Conference on Human-Computer-Interaction, 2019, pp. 30–41.

[13] A.D. Frederiks, J.R. Octavia, C. Vandevelde, J. Saldien, Towards participatory design of social robots, in: Proceedings of the IFIP Conference on Human–Computer Interaction, Springer, 2019, pp. 527–535.

[14] E. Efthimiou, S.-E. Fotinea, A. Vacalopoulou, X.S. Papageorgiou, A. Karavasili, T. Goulas, User centered design in practice: adapting HRI to real user needs, in: Proceedings of the 12th ACM International Conference on PErvasive Technologies Related to Assistive Environments, 2019, pp. 425–429.

[15] G. Fischer, D. Fogli, A. Piccinno, Revisiting and broadening the meta-design framework for end-user development, in: New Perspectives in End-User Development, Springer, 2017, pp. 61–97.

[16] F. Paternò, V. Wulf, New Perspectives in End-User Development, Springer, 2017.

[17] B.R. Barricelli, F. Cassano, D. Fogli, A. Piccinno, End-user development, end-user programming and end-user software engineering: a systematic mapping study, J. Syst. Softw. 149 (2019) 101–137.

[18] G. Fischer, E. Giaccardi, Y. Ye, A.G. Sutcliffe, N. Mehandjiev, Meta-design: a manifesto for end-user development, Commun. ACM 47 (9) (2004) 33–37.

[19] G. Fischer, End-user development: from creating technologies to transforming cultures, in: Proceedings of the International Symposium on End User Development, Springer, 2013, pp. 217–222.

[20] L. Baillie, C. Breazeal, P. Denman, M.E. Foster, K. Fischer, J.R. Cauchard, The challenges of working on social robots that collaborate with people, in: Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems, 2019, pp. 1–7.

[21] A.J. Ko, R. Abraham, L. Beckwith, A. Blackwell, M. Burnett, M. Erwig, C. Scaffidi, J. Lawrance, H. Lieberman, B. Myers, et al., The state of the art in end-user software engineering, ACM Comput. Surv. (CSUR) 43 (3) (2011) 1–44.

[22] S. Calinon, F. Guenter, A. Billard, On learning, representing, and generalizing a task in a humanoid robot, IEEE Trans. Syst. Man Cybern. Part B (Cybern.) 37 (2) (2007) 286–298, doi:10.1109/TSMCB.2006.886952.

[23] H. Friedrich, S. Münch, R. Dillmann, S. Bocionek, M. Sassin, Robot programming by demonstration (RPD): supporting the induction by human interaction, Mach. Learn. 23 (2) (1996) 163–189, doi:10.1007/BF00117443.

[24] J.F. Gorostiza, M.A. Salichs, Natural programming of a social robot by dialogs, Proceedings of the AAAI Fall Symposium: Dialog with Robots, 2010.

[25] P.P. Ray, A survey on visual programming languages in internet of things, Sci. Program. 2017 (2017), doi:10.1155/2017/1231430.

[26] D.-Q. Zhang, K. Zhang, On the design of a generic visual programming environment, in: Proceedings of the 1998 IEEE Symposium on Visual Languages, IEEE, 1998, pp. 88–89, doi:10.1109/VL.1998.706147.

[27] B.R. Barricelli, S. Valtolina, A visual language and interactive system for end-user development of internet of things ecosystems, J. Vis. Lang. Comput. 40 (2017) 1–19, doi:10.1016/j.jvlc.2017.01.004.

[28] P.E. Dickson, J.E. Block, G.N. Echevarria, K.C. Keenan, An experience-based comparison of unity and unreal for a stand-alone 3D game development course, in: Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education, ACM, 2017, pp. 70–75, doi:10.1145/3059009.3059013.

[29] I. Sagredo-Olivenza, P.P. Gómez-Martin, M.A. Gómez-Martin, P.A. González-Calero, Combining neural networks for controlling non-player characters in games, in: Proceedings of the International Work-Conference on Artificial Neural Networks, Springer, 2017, pp. 694–705, doi:10.1007/978-3-319-59147-6_59.

[30] R. Francese, M. Risi, G. Tortora, Iconic languages: towards end-user programming of mobile applications, J. Vis. Lang. Comput. 38 (2017) 1–8, doi:10.1016/j.jvlc.2016.10.009.

[31] J.M. Mota, I. Ruiz-Rube, J.M. Dodero, I. Arnedillo-Sánchez, Augmented reality mobile app development for all, Comput. Electr. Eng. 65 (2018) 250–260, doi:10.1016/j.compeleceng.2017.08.025.

[32] M.G. Maceli, Tools of the trade: a survey of technologies in end-user development literature, in: Proceedings of the International Symposium on End User Development, Springer, 2017, pp. 49–65.

[33] D. Tetteroo, P. Markopoulos, A review of research methods in end user development, in: Proceedings of the International Symposium on End User Development, Springer, 2015, pp. 58–75.

[34] F. Paternò, End user development: survey of an emerging field for empowering people, ISRN Softw. Eng. 2013 (2013), doi:10.1155/2013/532659.

[35] M. Santos, M.L.B. Villela, Characterizing end-user development solutions: a systematic literature review, in: Proceedings of the 2019 International Conference on Human-Computer Interaction, Springer, 2019, pp. 194–209.

[36] F. Patern, C. Santoro, End-user development for personalizing applications, things, and robots, Int. J. Hum. Comput. Stud. 131 (2019) 120–130, doi:10.1016/j.ijhcs.2019.06.002.

[37] A. Bellucci, A. Vianello, Y. Florack, L. Micallef, G. Jacucci, Augmenting objects at home through programmable sensor tokens: a design journey, Int. J. Hum. Comput. Stud. 122 (2019) 211–231, doi:10.1016/j.ijhcs.2018.09.002.

[38] K. Dill, Structural architecture-common tricks of the trade, Game AI Pro: Collected Wisdom of Game AI Professionals, CRC Press, 2013, p. 61.

[39] G.N. Yannakakis, J. Togelius, Artificial Intelligence and Games, 2, Springer, 2018, doi:10.1007/978-3-319-63519-4.

[40] M. Colledanchise, P. Ögren, Behavior Trees in Robotics and AI: An Introduction, CRC Press, 2018, doi:10.1201/9780429489105.

[41] A. Hentout, A. Maoudj, B. Bouzouia, A survey of development frameworks for robotics, in: Proceedings of the 8th International Conference on Modelling, Identification and Control (ICMIC), IEEE, 2016, pp. 67–72.

[42] D. Kortenkamp, R. Simmons, D. Brugali, Robotic systems architectures and programming, in: Springer Handbook of Robotics, Springer, 2016, pp. 283–306.

[43] F.A. Bravo, A.M. Gonzlez, E. Gonlez, A review of intuitive robot programming environments for educational purposes, in: Proceedings of the 3rd IEEE Colombian Conference on Automatic Control (CCAC), 2017, pp. 1–6, doi:10.1109/CCAC.2017.8276396.

[44] M.E. Karim, S. Lemaignan, F. Mondada, A review: can robots reshape k-12 STEM education? in: Proceedings of the 2015 IEEE International Workshop on Advanced Robotics and its Social Impacts (ARSO), 2015, pp. 1–8, doi:10.1109/ARSO.2015.7428217.

[45] D. Budgen, P. Brereton, Performing systematic literature reviews in software engineering, in: Proceedings of the 28th International Conference on Software Engineering, ACM, 2006, pp. 1051–1052.

[46] G. Golovchinsky, Cognitive Dimensions Analysis of Interfaces for Information Seeking, arXiv preprint arXiv:0908.3523(2009).

[47] T.R.G. Green, M. Petre, Usability analysis of visual programming environments: a 'cognitive dimensions' framework, J. Vis. Lang. Comput. 7 (2) (1996) 131–174, doi:10.1006/jvlc.1996.0009.

[48] J. Dagit, J. Lawrance, C. Neumann, M. Burnett, R. Metoyer, S. Adams, Using cognitive dimensions: advice from the trenches, J. Vis. Lang. Comput. 17 (4) (2006) 302–327.

[49] T. Green, A. Blackwell, Cognitive dimensions of information artefacts: a tutorial, Proceedings of the 1998 BCS HCI Conference, volume 98, 1998.

[50] J. Diprose, B. MacDonald, J. Hosking, B. Plimmer, Designing an API at an appropriate abstraction level for programming social robot applications, J. Vis. Lang. Comput. 39 (2017) 22–40, doi:10.1016/j.jvlc.2016.07.005.

[51] A. Carfì, J. Villalobos, E. Coronado, B. Bruno, F. Mastrogiovanni, Can human-inspired learning behaviour facilitate human–robot interaction? Int. J. Soc. Robot. 1 (2019) 1–14, doi:10.1007/s12369-019-00548-5.

[52] F. Steinmetz, A. Wollschläger, R. Weitschat, Razer – a human-robot interface for visual task-level programming and intuitive skill parametrization, IEEE Rob. Autom. Lett. 3 (3) (2018) 1362–1369.

[53] C. Paxton, A. Hundt, F. Jonathan, K. Guerin, G.D. Hager, CoSTAR: instructing collaborative robots with behavior trees and vision, in: Proceedings of the 2017 IEEE International Conference on Robotics and Automation (ICRA), 2017, pp. 564–571, doi:10.1109/ICRA.2017.7989070.

[54] D. Weintrop, D.C. Shepherd, P. Francis, D. Franklin, Blockly goes to work: block-based programming for industrial robots, in: Proceedings of the 2017 IEEE Blocks and Beyond Workshop (BB), 2017, pp. 29–36, doi:10.1109/BLOCKS.2017.8120406.

[55] E. Bilotta, P. Pantano, Some problems of programming in robotics, in: Proceedings of the 12th Annual Workshop of the Psychology of Programming Interest Group, Psychology of Programming Interest Group, 2000, pp. 209–220.

[56] G. Serafini, Teaching programming at primary schools: visions, experiences, and long-term research prospects, in: I. Kalaš, R.T. Mittermeir (Eds.), Informatics in Schools. Contributing to 21st Century Education, Springer, Berlin, Heidelberg, 2011, pp. 143–154, doi:10.1007/978-3-642-24722-4_13.

[57] D. Weintrop, U. Wilensky, To block or not to block, that is the question: students' perceptions of blocks-based programming, in: Proceedings of the 14th International Conference on Interaction Design and Children, 2015, pp. 199–208, doi:10.1145/2771839.2771860.

[58] D.S. Touretzky, C. Gardner-McCune, Calypso for cozmo: robotic AI for everyone (abstract only), Proceedings of the 49th ACM Technical Symposium on Computer Science Education, 2018, doi:10.1145/3159450.3162200. 1110–1110

[59] J. Shin, R. Siegwart, S. Magnenat, Visual programming language for thymio II robot, Proceedings of the Conference on Interaction Design and Children (IDC'14), ETH Zürich, 2014.

[60] M.F. Costabile, D. Fogli, G. Fresta, P. Mussio, A. Piccinno, Software environments for end-user development and tailoring, PsychNol. J. 2 (1) (2004) 99–122.

[61] I. Zubrycki, M. Kolesiński, G. Granosik, Graphical programming interface for enabling non-technical professionals to program robots and internet-of-things devices, in: Proceedings of the International Work-Conference on Artificial Neural Networks, Springer, 2017, pp. 620–631.

[62] E.I. Barakova, J.C.C. Gillesen, B.E.B.M. Huskens, T. Lourens, End-user programming architecture facilitates the uptake of robots in social therapies, Rob. Auton. Syst. 61 (7) (2013) 704–713, doi:10.1016/j.robot.2012.08.001.

[63] T. Fong, I. Nourbakhsh, K. Dautenhahn, A survey of socially interactive robots, Rob. Auton. Syst. 42 (3) (2003) 143–166, doi:10.1016/S0921-8890(02)00372-X. Socially Interactive Robots

[64] E. Coronado, F. Mastrogiovanni, G. Venture, Design of a human-centered robot framework for end-user programming and applications, in: Proceedings of the 2019 Robot Design, Dynamics and Control, ROMANSY 22, Springer, 2019, pp. 450–457, doi:10.1007/978-3-319-78963-7_56.

[65] Y. Oishi, T. Kanda, M. Kanbara, S. Satake, N. Hagita, Toward end-user programming for robots in stores, in: Proceedings of the Companion of the 2017 ACM/IEEE International Conference on Human-Robot Interaction, ACM, 2017, pp. 233–234, doi:10.1145/3029798.3038340.

[66] F. Corno, L. De Russis, A. Monge Roffarello, My IoT puzzle: debugging IF-THEN rules through the jigsaw metaphor, in: Proceedings of the 2019 End-User Development, Springer International Publishing, Cham, 2019, pp. 18–33, doi:10.1007/978-3-030-24781-2_2.

[67] N. Leonardi, M. Manca, F. Paternò, C. Santoro, Trigger-action programming for personalising humanoid robot behaviour, in: Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems, ACM, 2019, p. 445, doi:10.1145/3290605.3300675.

[68] F. Corno, L. De Russis, A. Monge Roffarello, Empowering end users in debugging trigger-action rules, in: Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems, ACM, 2019, p. 388, doi:10.1145/3290605.3300618.

[69] S. Gaudl, Building Robust Real-Time Game AI: Simplifying & Automating Integral Process Steps in Multi-Platform Design, University of Bath, 2016 Ph.D. thesis.

[70] M. Colledanchise, P. Ögren, How behavior trees modularize hybrid control systems and generalize sequential behavior compositions, the subsumption architecture, and decision trees, IEEE Trans. Rob. 33 (2) (2017) 372–389, doi:10.1109/TRO.2016.2633567.

[71] J. Bohren, S. Cousins, The SMACH high-level executive [ROS news], IEEE Robot. Autom. Mag. 17 (4) (2010) 18–20, doi:10.1109/MRA.2010.938836.

[72] D.D. Hils, Visual languages and computing survey: data flow visual programming languages, J. Vis. Lang. Comput. 3 (1) (1992) 69–101, doi:10.1016/1045-926X(92)90034-J.

[73] D. Weintrop, Block-based programming in computer science education, Commun. ACM 62 (8) (2019) 22–25, doi:10.1145/3341221.

[74] D. Weintrop, U. Wilensky, Comparing block-based and text-based programming in high school computer science classrooms, ACM Trans. Comput. Educ. (TOCE) 18 (1) (2017) 1–25, doi:10.1145/3089799.

[75] M. Resnick, J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. Silver, B. Silverman, et al., Scratch: programming for all, Commun. ACM 52 (11) (2009) 60–67, doi:10.1145/1592761.1592779.

[76] D. Garcia, L. Segars, J. Paley, Snap!(build your own blocks): tutorial presentation, J. Comput. Sci. Coll. 27 (4) (2012) 120–121.

[77] E. Pasternak, R. Fenichel, A.N. Marshall, Tips for creating a block language with blockly, in: Proceedings of the 2017 IEEE Blocks and Beyond Workshop (B&B), IEEE, 2017, pp. 21–24.

[78] Google Blockly. https://developers.google.com/blockly/, 2018, (accessed 15 August 2019).

[79] B. Kitchenham, Procedures for Performing Systematic Reviews, Keele, UK, Keele University 33 (2004) 1–26.

[80] T. Schulz, J. Torresen, J. Herstad, Animation techniques in human-robot interaction user studies: a systematic literature review, ACM Trans. Human-Robot Interact. (THRI) 8 (2) (2019) 1–22, doi:10.1145/3317325.

[81] A.R. Martin, S. Colton, Towards liveness in game development, in: Proceedings of the 2019 IEEE Conference on Games (CoG), IEEE, 2019, pp. 1–4.

[82] P. Rein, S. Ramson, J. Lincke, R. Hirschfeld, T. Pape, Exploratory and Live, Programming and Coding: A Literature Study Comparing Perspectives on Liveness, arXiv preprint arXiv:1807.08578(2018).

[83] M. Campusano, J. Fabry, Live robot programming: the language, its implementation, and robot API independence, Sci. Comput. Program. 133 (2017) 1–19.

[84] S.L. Tanimoto, Viva: a visual language for image processing, J. Vis. Lang. Comput. 1 (2) (1990) 127–139.

[85] S. Anjomshoae, A. Najjar, D. Calvaresi, K. Främling, Explainable agents and robots: results from a systematic literature review, in: Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems, International Foundation for Autonomous Agents and Multiagent Systems, 2019, pp. 1078–1088.

[86] L.D. Riek, Wizard of Oz studies in HRI: a systematic review and new reporting guidelines, J. Human-Robot Interact. 1 (1) (2012) 119–136.

[87] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A.Y. Ng, ROS: an open-source robot operating system, in: Proceedings of the ICRA Workshop on Open Source Software, 3, Kobe, 2009, p. 5.

[88] E. Pot, J. Monceaux, R. Gelin, B. Maisonnier, Choregraphe: a graphical tool for humanoid robot programming, in: Proceedings of the 18th IEEE International Symposium on Robot and Human Interactive Communication, RO-MAN 2009, 2009, pp. 46–51, doi:10.1109/ROMAN.2009.5326209.

[89] Seobook, Keyword density analizer. http://tools.seobook.com/general/keyword-density/, 2003, (accessed 15 August 2019).

[90] H. Zhang, M.A. Babar, P. Tell, Identifying relevant studies in software engineering, Inf. Softw. Technol. 53 (6) (2011) 625–637.

[91] D.F. Glas, T. Kanda, H. Ishiguro, Human-robot interaction design using interaction composer eight years of lessons learned, in: Proceedings of the 11th ACM/IEEE International Conference on Human-Robot Interaction (HRI), 2016, pp. 303–310, doi:10.1109/HRI.2016.7451766.

[92] D. Glas, S. Satake, T. Kanda, N. Hagita, An interaction design framework for social robots, in: Proceedings of the 2012 Robotics: Science and Systems, 7, 2012, p. 89.

[93] K. Petersen, S. Vakkalanka, L. Kuzniarz, Guidelines for conducting systematic mapping studies in software engineering: an update, Inf. Softw. Technol. 64 (2015) 1–18.

[94] S. Keele, et al., Guidelines for Performing Systematic Literature Reviews in Software Engineering, Technical Report, EBSE, 2007. Technical report, Ver. 2.3 EBSE Technical Report

[95] P. Brereton, B.A. Kitchenham, D. Budgen, M. Turner, M. Khalil, Lessons from applying the systematic literature review process within the software engineering domain, J. Syst. Softw. 80 (4) (2007) 571–583.

[96] J. Jackson, Microsoft robotics studio: a technical introduction, IEEE Robot. Autom. Mag. 14 (4) (2007) 82–87, doi:10.1109/M-RA.2007.905745.

[97] CodeIt!. https://github.com/hcrlab/code_it, 2018, (accessed 15 August 2019).

[98] B. Jost, M. Ketterl, R. Budde, T. Leimbach, Graphical programming environments for educational robots: open roberta – yet another one? in: Proceedings of the 2014 IEEE International Symposium on Multimedia, 2014, pp. 381–386, doi:10.1109/ISM.2014.24.

[99] I. Zubrycki, G. Granosik, Designing an interactive device for sensory therapy, in: Proceedings of the 11th ACM/IEEE International Conference on Human-Robot Interaction (HRI), 2016, pp. 545–546, doi:10.1109/HRI.2016.7451848.

[100] M. Seraj, S. Autexier, J. Janssen, Beesm, a block-based educational programming tool for end users, in: Proceedings of the 10th Nordic Conference on Human-Computer Interaction, NordiCHI '18, ACM, 2018, pp. 886–891, doi:10.1145/3240167.3240239.

[101] RIZE. https://enriquecoronadozu.github.io/RIZE/, 2020 (accessed 20 February 2020).

[102] P. Ziafati, F. Lera, A. Costa, A. Nazarikhorram, L. Van Der Torre, A. Nazarikhor, Procrob architecture for personalized social robotics, in: Proceedings of the Robots for Learning Workshop@ HRI, 2017, pp. 6–9.

[103] C. Datta, B.A. MacDonald, Architecture of an extensible visual programming environment for authoring behaviour of personal service robots, in: Proceedings of the First IEEE International Conference on Robotic Computing (IRC), 2017, pp. 156–159, doi:10.1109/IRC.2017.60.

[104] F. Erich, M. Hirokawa, K. Suzuki, A visual environment for reactive robot programming of macro-level behaviors, in: Proceedings of the Social Robotics, Springer, 2017, pp. 577–586, doi:10.1007/978-3-319-70022-9_57.

[105] D. Porfirio, A. Sauppé, A. Albarghouthi, B. Mutlu, Authoring and verifying human-robot interactions, in: Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology, UIST '18, ACM, 2018, pp. 75–86, doi:10.1145/3242587.3242634.

[106] A. Sauppé, B. Mutlu, Design patterns for exploring and prototyping human-robot interactions, in: Proceedings of the 32nd Annual ACM Conference on Human Factors in Computing Systems, CHI '14, ACM, 2014, pp. 1439–1448, doi:10.1145/2556288.2557057.

[107] N. Buchina, S. Kamel, E. Barakova, Design and evaluation of an end-user friendly tool for robot programming, in: Proceedings of the 25th IEEE International Symposium on Robot and Human Interactive Communication (RO–MAN), IEEE, 2016, pp. 185–191.

[108] QT. https://www.qt.io/), 2019, (accessed 15 August 2019).

[109] M.H. Lee, H.S. Ahn, K. Wang, B. MacDonald, Design of an API for integrating robotic software frameworks, in: Proceedings of the 2014 Australasian Conference on Robotics and Automation (ACRA 2014), 2, 2014, p. 1.

[110] H. Bruyninckx, Open robot control software: the OROCOS project, in: Proceedings of the 2001 ICRA IEEE International Conference on Robotics and Automation (Cat. No. 01CH37164), 3, 2001, pp. 2523–2528 vol.3, doi:10.1109/ROBOT.2001.933002.

[111] C. Datta, H.Y. Yang, I. Kuo, E. Broadbent, B.A. MacDonald, Software platform design for personal service robots in healthcare, in: Proceedings of the 6th IEEE Conference on Robotics, Automation and Mechatronics (RAM), 2013, pp. 156–161, doi:10.1109/RAM.2013.6758576.

[112] S. Tilkov, S. Vinoski, Node.js: using javascript to build high-performance network programs, IEEE Internet Comput. 14 (6) (2010) 80–83, doi:10.1109/MIC.2010.145.

[113] E. Bainomugisha, A.L. Carreton, T.v. Cutsem, S. Mostinckx, W.d. Meuter, A survey on reactive programming, ACM Comput. Surv. (CSUR) 45 (4) (2013) 52, doi:10.1145/2501654.2501666.

[114] RRP. https://github.com/FlorisE/RRP, 2018, (accessed 15 August 2019).

[115] M. Kwiatkowska, G. Norman, D. Parker, Prism 4.0: verification of probabilistic real-time systems, in: Proceedings of the 2011 Computer Aided Verification, Springer, Berlin, Heidelberg, 2011, pp. 585–591, doi:10.1007/978-3-642-22110-1_47.

[116] RoVer. https://github.com/Wisc-HCI/RoVer, 2018, (accessed 15 August 2019).

[117] G. Metta, P. Fitzpatrick, L. Natale, Yarp: yet another robot platform, Int. J. Adv. Rob. Syst. 3 (1) (2006) 8, doi:10.5772/5761.

[118] A. Haddadi, K. Sundermeyer, in: Foundations of Distributed Artificial Intelligence, John Wiley & Sons, Inc., 1996, pp. 169–185.

[119] C. Crick, G. Jay, S. Osentoski, B. Pitzer, O.C. Jenkins, Rosbridge: ROS for Non-ROS Users, Springer, 2017, pp. 493–504.

[120] J. Huang, T. Lau, M. Cakmak, Design and evaluation of a rapid programming system for service robots, in: Proceedings of the 11th ACM/IEEE International Conference on Human-Robot Interaction (HRI), 2016, pp. 295–302, doi:10.1109/HRI.2016.7451765.

[121] V. Paramasivam, J. Huang, S. Elliott, M. Cakmak, Computer science outreach with end-user robot-programming tools, in: Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education, SIGCSE '17, ACM, 2017, pp. 447–452, doi:10.1145/3017680.3017796.

[122] B. Wulff, A. Wilson, B. Jost, M. Ketterl, An adopter centric API and visual programming interface for the definition of strategies for automated camera tracking, in: Proceedings of the 2015 IEEE International Symposium on Multimedia (ISM), 2015, pp. 587–592, doi:10.1109/ISM.2015.106.

[123] Snap. https://snap.berkeley.edu/), 2018, (accessed 15 August 2019).

[124] ZeroMQ. http://zeromq.org/, 2020, (accessed 20 February 2020).

[125] Nanomsg. https://nanomsg.org/, 2020, (accessed 20 February 2020).

[126] E. Coronado, X. Indurkhya, G. Venture, Robots meet children, development of semi-autonomous control systems for children-robot interaction in the wild, Proceedings of the IEEE International Conference on Advanced Robotics and Mechatronics (ICARM), IEEE, 2019.

[127] X. Indurkhya, I. Takamune, E. Coronado, P. Zguda, B. Indurkhya, G. Venture, Creating a robust vocalization-based protocol for analyzing CRI group studies in the wild, Proceedings of the International Conference on Human Robot Interaction, ACM/IEEE, 2018.

[128] L. Rincon, E. Coronado, H. Hendra, J. Phan, Z. Zainalkefli, G. Venture, Expressive states with a robot arm using adaptive fuzzy and robust predictive controllers, in: Proceedings of the 3rd International Conference on Control and Robotics Engineering (ICCRE), 2018, pp. 11–15, doi:10.1109/ICCRE.2018.8376425.

[129] J. Forcier, P. Bissex, W.J. Chun, Python Web Development with Django, Addison-Wesley Professional, 2008.

[130] S. Vinoski, Server-sent events with yaws, IEEE Internet Comput. 16 (5) (2012) 98–102.

[131] T.J.M. Bench-Capon, Knowledge Representation: An Approach to Artificial Intelligence, 32, Elsevier, 2014.

[132] D. Mark, AI architectures: a culinary guide, Game Dev. Mag. 19 (8) (2012) 7–12.

[133] B. Ur, E. McManus, M. Pak Yong Ho, M.L. Littman, Practical trigger-action programming in the smart home, in: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, 2014, pp. 803–812.

[134] M. Dawe, S. Garolinski, L. Dicken, T. Humphreys, D. Mark, Behavior selection algorithms: an overview, Game AI Pro: Collected Wisdom of Game AI Professionals, CRC Press, 2014, pp. 47–60.

[135] E.W. Dijkstra, Letters to the editor: go to statement considered harmful, Commun. ACM 11 (3) (1968) 147–148.

[136] P. Harwood, Multi Modal Human Robot Interaction Interface, Ecole Central of Nantes, 2016, 2016 Master's thesis.

[137] L. Joseph, J. Cacace, Mastering ROS for Robotics Programming: Design, Build, and Simulate Complex Robots Using the Robot Operating System, Packt Publishing Ltd, 2018.

[138] TiViPE. http://www.tivipe.com/TVPeducation/TVPuse.pdf, 2019 (accessed 25 August 2019).

[139] K. Finstad, The system usability scale and non-native english speakers, J. Usabil. Stud. 1 (4) (2006) 185–188.

[140] S.G. Hart, Nasa-task load index (NASA-TLX); 20 years later, in: Proceedings of the 2006 Human Factors and Ergonomics Society Annual Meeting, 50, Sage Publications, CA: Los Angeles, 2006, pp. 904–908.

[141] C. Ebert, J. Cain, Cyclomatic complexity, IEEE Softw. 33 (6) (2016) 27–29.

[142] R. Bischoff, T. Guhl, E. Prassler, W. Nowak, G. Kraetzschmar, H. Bruyninckx, P. Soetens, M. Haegele, A. Pott, P. Breedveld, et al., Brics-best practice in robotics, in: Proceedings of the 41st International Symposium on Robotics (ISR) and 2010 6th German Conference on Robotics (ROBOTIK), VDE, 2010, pp. 1–8.

[143] R. Rajkumar, M. Gagliardi, L. Sha, The real-time publisher/subscriber interprocess communication model for distributed real-time systems: design and implementation, in: Proceedings of the 1995 Real-Time Technology and Applications Symposium, 1995, pp. 66–75, doi:10.1109/RTTAS.1995.516203.

[144] A.S. Tanenbaum, M. Van Steen, Distributed Systems: Principles and Paradigms, Prentice-Hall, 2007.

[145] A. Orebäck, H.I. Christensen, Evaluation of architectures for mobile robotics, Auton. Robot. 14 (1) (2003) 33–49, doi:10.1023/A:1020975419546.

[146] W. Gellerich, M. Kosiol, E. Ploedereder, Where does GOTO go to? in: Proceedings of the International Conference on Reliable Software Technologies, Springer, 1996, pp. 385–395.

[147] P. Janssen, Visual Dataflow Modelling-Some Thoughts on Complexity, Proceedings of the 32nd eCAADe Conference, England, UK, 10-12 September 2014, pp. 547-556.

[148] E. Coronado, F. Mastrogiovanni, G. Venture, Development of intelligent behaviors for social robots via user-friendly and modular programming tools, in: Proceedings of the 2018 IEEE Workshop on Advanced Robotics and its Social Impacts (ARSO), 2018, pp. 62–68, doi:10.1109/ARSO.2018.8625839.

[149] S. Wang, H.I. Christensen, Tritonbot: first lessons learned from deployment of a long-term autonomy tour guide robot, in: Proceedings of the 27th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN), 2018, pp. 158–165, doi:10.1109/ROMAN.2018.8525845.

[150] I. Leite, C. Martinho, A. Paiva, Social robots for long-term interaction: a survey, Int. J. Soc. Robot. 5 (2) (2013) 291–308, doi:10.1007/s12369-013-0178-y.

[151] A.F. Blackwell, C. Britton, A. Cox, T.R.G. Green, C. Gurr, G. Kadoda, M.S. Kutar, M. Loomes, C.L. Nehaniv, M. Petre, C. Roast, C. Roe, A. Wong, R.M. Young, Cognitive dimensions of notations: design tools for cognitive technology, in: Cognitive Technology: Instruments of Mind, Springer, Berlin, Heidelberg, 2001, pp. 325–341, doi:10.1007/3-540-44617-6_31.

[152] D. Moody, The "physics" of notations: toward a scientific basis for constructing visual notations in software engineering, IEEE Trans. Softw. Eng. 35 (6) (2009) 756–779.

[153] A. Assila, H. Ezzedine, et al., Standardized usability questionnaires: features and quality focus, Electron. J. Comput. Sci. Inf. Technol. eJCIST 6 (1) (2016).

[154] J. Brooke, et al., Sus-a quick and dirty usability scale, Usabil. Eval. Ind. 189 (194) (1996) 4–7.

[155] , Development of NASA-TLX (task load index): results of empirical and theoretical research, in: P.A. Hancock, N. Meshkati (Eds.), Human Mental Workload, Advances in Psychology, 52, North-Holland, 1988, pp. 139–183, doi:10.1016/S0166-4115(08)62386-9.

[156] A.P.O.S. Vermeeren, E.L.-C. Law, V. Roto, M. Obrist, J. Hoonhout, K. Väänänen-Vainio-Mattila, User experience evaluation methods: current state and development needs, in: Proceedings of the 6th Nordic Conference on Human-Computer Interaction: Extending Boundaries, NordiCHI '10, ACM, 2010, pp. 521–530, doi:10.1145/1868914.1868973.

[157] User Experience Basics. https://www.usability.gov/what-and-why/user-experience.html, 2018 (accessed 15 August 2019).
[158] F. Mastrogiovanni, A. Sgorbissa, R. Zaccaria, A system for hierarchical planning in service mobile robotics, Proceedings of the Eight Conference on Intelligent Autonomous Systems (IAS-8), 2004. Amsterdam, The Netherlands
[159] A. Capitanelli, M. Maratea, F. Mastrogiovanni, M. Vallati, On the manipulation of articulated objects in human-robot cooperation scenarios, Rob. Auton. Syst. 109 (2018) 139–155, doi:10.1016/j.robot.2018.08.003.