Received December 10, 2020, accepted December 28, 2020, date of publication January 14, 2021,
date of current version January 22, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3051605

Towards a Modular and Distributed End-User Development Framework for Human-Robot Interaction

ENRIQUE CORONADO[1], DOMINIQUE DEUFF[2], PAMELA CARRENO-MEDRANO[3], LEIMIN TIAN[3], DANA KULIĆ[3], (Member, IEEE), SHANTI SUMARTOJO[4], FULVIO MASTROGIOVANNI[5], AND GENTIANE VENTURE[1], (Senior Member, IEEE)

[1]Department of Mechanical System Engineering, Tokyo University of Agriculture and Technology, Tokyo 184-0012, Japan
[2]Orange Lab, 22300 Lannion, France
[3]Faculty of Engineering, Monash University, Clayton, VIC 3800, Australia
[4]Faculty of Art, Design and Architecture, Monash University, Caulfield East, VIC 3145, Australia
[5]Department of Informatics, Bioengineering, Robotics and Systems Engineering, University of Genoa, 16145 Genoa, Italy

Corresponding author: Enrique Coronado (enriquecoronadozu@gmail.com)

This work was supported by the Institute for Global Innovation Research, Tokyo University of Agriculture and Technology.

ABSTRACT In an effort towards the democratization of Robotics, this article presents a novel End-User Development framework called *Robot Interfaces From Zero Experience* (RIZE). The framework provides a set of useful software tools for the creation of robot-oriented software architectures and programming interfaces, as well as the modeling and execution of robot behaviors, with a specific emphasis on social behaviors. Programming interfaces built on top of RIZE enable professionals with different backgrounds and interests to design, adapt, and scale-up robotics applications. As an example of a programming interface, we present Open RIZE, which exploits an End-User Programming paradigm combining blocks, tables, and forms-filling interfaces. Unlike previous approaches, robot behavioral code generated by Open RIZE is intrinsically modular, re-usable, scalable, neutral to the employed programming language, and platform-agnostic. In the article, we present the main design guidelines and features of Open RIZE. Additionally, we perform an initial usability evaluation of the Open RIZE interface in an online workshop. Preliminary results using the System Usability Scale with 10 novice end-users indicate that Open RIZE is easy-to-use and learn.

INDEX TERMS Robotics, human-robot interaction, end-user development, robot programming, social robots, service robots.

I. INTRODUCTION

Human-Robot Interaction (HRI) is a field populated to a great extent by researchers and professionals expert in Information Technology (IT) [1]. In the literature, it is common to read about HRI studies that are carried out by such technically skilled people, who are often referred to as *high-tech scribes* [2], in various controlled scenarios [3]. However, a recent call for the creation of more robust and usable robot-based applications suggests that it is time to transfer Robotics research from the laboratories and industries to natural, every-day environments [3].

"In the wild" HRI experiments [4] often require teaming with social scientists or interaction designers, and in general with end-users to model realistic and meaningful scenarios [5], [6]. Two approaches advocating the inclusion of non-roboticists or novice end-users in the design of robot-based applications are the *user-centered* [7] and the *participatory design* [8] perspectives. As described in [9], [10], these methods tend to adopt a "design before using" approach. Two issues affect their applicability, namely (i) the possible misunderstandings between expert roboticists and non-technically skilled researchers when interpreting the HRI contexts or application requirements, and (ii) the diversity of those end-users whose background may be in social sciences [10].

End-User Development (EUD) is an emergent human-centered approach and a suitable companion to user-centered and participatory design [10]. It is considered a socio-technical activity [10], where problem owners, i.e., end-users,

The associate editor coordinating the review of this manuscript and approving it for publication was Giuseppe Desolda.

become independent from high-tech scribes [2]. End-User Programming (EUP) is a sub-area of EUD which only focuses on the program coding phase. EUD also includes those methods supporting the entire software development life-cycle, i.e., installation, configuration, re-design, debugging, scaling, and execution [11].

This article presents (i) the *Robot Interfaces from Zero Experience* (RIZE) framework, and (ii) an EUD/EUP tool called *Open RIZE*. On the one hand, the RIZE framework proposes a general software architecture and a set of software tools for the creation of EUD/EUP programming environments for Robotics systems. On the other hand, Open RIZE is an experimental programming environment built on top of the RIZE tools for enabling social researchers to create real-world HRI applications with social and service robots. Open RIZE can be installed as a typical Windows and OSX application using Wizards. Detailed instructions on how to install, use, and modify Open RIZE (including Linux-bases systems) are available in [12].

The article is organized as follows. Section II presents related work and summarizes our contributions. Section III presents an overview of the RIZE framework. Section IV describes the main features of Open RIZE, whereas Section V describes the cognitive software architecture implemented in Open RIZE. Section VI briefly describes a set of applications developed using different versions of Open RIZE and presents results from an initial usability assessment. Conclusions and future work follow.

II. RELATED WORK AND CONTRIBUTIONS

According to [13], there are three types of EUD/EUP tools for Robotics, namely educational, industrial, and professional tools, the main differences between these categories being their objectives, type of tasks and scenarios [13]. Relevant examples of educational tools are LearnBlock [14] and OpenRoberta [15]. These two interfaces are compatible with different types of teaching approaches and toy robots, and have been designed with two objectives in mind: the first is the development of critical thinking skills in kids through solving ''toy problems'', i.e., illustrative exercises or puzzles [16], [17], whereas the second is enabling an effective yet gentle transition to a general-purpose programming language such as Python or C++. Students can meet these goals after several software development courses, typically provided by educational institutions.

The main objectives of EUD/EUP industrial tools are reducing training efforts and increasing the flexibility of robot-based systems. EUD/EUP tools in this category enable factory workers to intuitively re-program robots without using nor learning a general-purpose programming language [1]. EUP approaches often used for this task are Visual Programming [18], [19] and Programming by Demonstration (PbD) [20], [21]. In many industrial cases, robots are deployed in scenarios whereby a direct interaction with a robot is not required [22], [23]. However, a number of emergent industrial scenarios focus on enabling the collaboration between humans and robots [24], [25].

The focus of professional EUD/EUP tools is bridging the gap between robots and their adoption in novel, real-world settings beyond educational and industrial scenarios. Tools used for this aim must be able to provide means to leverage the social and emotional capabilities that robots can exhibit and work ''in the wild'', as well as enable the development of explainable, useful, usable, and enjoyable HRI applications [13]. The majority of professional EUD/EUP tools end-users, such as therapists, interaction designers, sellers, and psychologists, lack the time and interest to learn advanced IT topics [10]. Professional EUD/EUP tools should provide suitable programming abstraction levels enabling an intuitive modeling of HRI use cases, including the use of robot social skills [26]. Due to its modular and distributed architecture, the RIZE framework can be adapted for enabling the development of educational, industrial, and professional EUP/EUD environments. However, the interface developed to demonstrate the capabilities of this framework, denoted as Open RIZE, is mostly aimed to be an easy-to-use tool that enables different types of adult end-users the rapid prototyping of real-world HRI applications. Therefore, the rest of the article will center in EUD/EUP tools for proffesional use.

HRI processes (and the interaction with *virtual* agents), including those involving social capabilities, are typically designed by expert developers using general-purpose scripting languages such as Python. This approach is error-prone and generates accessibility, usability, and maintainability issues [27]. A recent trend towards improving the accessibility of scripting approaches is to enable the adoption of more user-friendly and engaging programming environments through the use of block-based programming languages such as Snap! [28], Scratch [29] and Google Blockly [30]. Figure 1 shows an example of a typical block-based programming environment in Google Blockly. Users can drag-and-drop a set of visual elements from a *toolbox* (left) to a *workspace* (middle). Then, the programming environment generates code in a selected general-purpose programming language for its subsequent execution. An advantage of this graphical approach compared to text-based programming environments is the reduction of syntax errors. These features

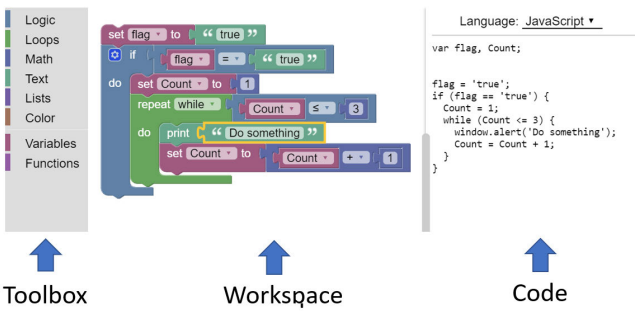


FIGURE 1. Example of a block-based programming environment in Google Blockly [30].

makes tools such as Scratch and Google Blockly popular solutions for creating educational EUP environments able to get children acquainted with different programming-related topics. However, some researchers have also proposed the use of block-based visual languages to enable professionals (rather than kids) to develop programs for social or service robots [31]–[33]. The programming approaches used in these professional-oriented block-based tools do not significantly differ from those used for educational purposes. Therefore, end-users require the use and the understanding of statements often used in programming languages, such as control loops, events, functions, and mathematical operators. They also need to build every aspect that will define the robot application, from data acquisition from sensors and its posterior processing to mechanisms that guide the decision-making and definition of low-level outputs of actuators (e.g., velocity and position of motors). As explained in [26], this low-level and general approach, which is presented in most EUP tools for robotics, produces a set of usability and maintenance issues. To this aim, programming tools for social and service robots must avoid the use of many low-level programming primitives. Instead, they must provide a set of reusable and easy to understand building blocks (denoted *primitives*) representing social skills which end-user can easily understand, combine and re-use [34]. These social skills are typically related to text-to-speech, animations, gestures, and the use of gaze. Moreover, approaches enabling the management of the defined robot behaviors, described in [26] as control primitives, must be transparent to users. Some of the most relevant approaches to control and model robot behaviors in EUP for robotics are described below.

In order to allow the rapid and high-quality development of scenarios where robots can socially interact with humans, professionals and designers prefer the use of Authoring Artificial Intelligence (AAI) methods over classical scripting and the use of programming primitives [35]. Popular AAI approaches used in consumer-oriented and industrial scenarios can be identified as rule-based systems, Finite State Machines (FSM), or Behaviors Trees (BTs).

Rule-based systems are the first AAI approach used to replace general-purpose, code-based behavioral development for the creation of interactive and social robots, as well as virtual agents [35]. While rule-based systems are still popular in certain fields, such as the Internet of Things (IoT) [36], [37], in Robotics, they have been replaced to a great extent FSMs and BTs [35], [38].

FSMs are structured methods for modeling the control flow of intelligent systems, including robots, which add the notions of states and transitions to rules-based systems. FSMs can be represented as directed graphs where each node of the graph represents a state. Transitions between states occur when state-specific conditions are met. A transition to a new state may trigger the execution of a sequence of robot behaviors. FSMs are relatively easy to learn and use, even for novice users [39], and enable the creation of robust collections of behaviors. Therefore, FSMs are often used to back-end EUP tools using data-flow visual languages [13]. However, practice suggests that the code based on FSMs and other data-flow methods tends to be messy. The resulting visual programs are difficult to analyze, re-use, maintain and scale [13], [38]. The most relevant EUD platforms for social robots using classical scripting notations (in block-based interfaces), rules, and data-flow visual languages, have been described in [13].

BTs are a more recent alternative to rule-based systems and FSMs. BTs enable the design and development of modular, complex, robust, and reactive behaviors [27]. BTs are quite popular tools for modeling the behavior of *non-player characters* in video games. However, only in the past few years have BTs been introduced in Robotics [38]. Implementation examples of BTs in Robotics for academic and industrial scenarios have been reported in [18], [38], [40]. They can be represented as directed trees, and are executed using a ''depth-first'' traversal procedure. Nodes in a BT can be of two types: (i) *composite nodes* or operators, which control the execution flow in the tree, and (ii) *leaf nodes*, which define the tasks composing the behaviors as well as the conditions triggering their execution. The BT execution algorithm assigns a status to each active node, which can be *success* if the task is completed without errors, or *failure* otherwise, or *running* during task execution. Composite nodes use these three possible statuses to perform behavior composition. For a deeper explanation of BTs the reader is referred to [38], [41]. Compared with FSM, BTs are known to be more modular: portions of code, denoted as ''sub-BTs'', can be easily re-used to create more complex tasks.

The main objective of RIZE is to provide a set of conceptual and operative tools that facilitate the integration of robot behavioral modules, specifically robot social skills, organized in a full-fledged software architecture. Rather than only focus on some specific programming or behavior modeling approach, RIZE was designed to enable the exploration of novel, structured, modular, reactive and advanced behavioral modeling methods and visual languages. This perspective represents the main difference with respect to previous EUP/EUD tools for robotics [1]. In the next sections, we present an example of a programming interface built on top of RIZE framework denoted as Open RIZE which uses a combination of FSMs and BTs for enabling decision-making. This interface integrates a programming environment extending the functionalities of Google Blockly to enable the intuitive creation of robot behaviors by novice end-users as well as the rapid prototyping and integration of robot-based applications.

Most approaches in the literature employing a block-based Visual Programming Language (VPL), such as Google Blockly, allow for the direct translation of block definitions to code in a text-based programming language. We explored this approach in an early prototype of RIZE [42]. The programming environment proposed in [42] translates the structure developed in Google Blockly to text-based code in Python. The resulting Python code consists of a set of robot-agnostic functions, i.e., those able to connect and be used with

different robots and to abstract low-level details and hardware instances [14]. These functions are executed by a Python server which is in charge of connecting perceptual, cognitive, knowledge representation, control, and social behavioral modules. The structure enables an easy re-use of modules developed in other programming languages.

Similar approaches targeted at robot-agnosticism have been proposed in professional [31], [43] and educational tools [14], [44]. However, block-based programming environments often limit the visualization of all graphical elements composing a program into a single view. Therefore, to find an specific behavior in complex programs that are composed of many block can be a cumbersome task. This issue, together with the lack of usability and maintainability inherited from scripting-based methods, hinders the management, scaling, reuse, update, and testing of robot behaviors. While these aspects can be trivial when designing interaction processes in simple scenarios and using educational tools, they become relevant for developing real-world and complex HRI applications requiring the use of many visual elements, as it has been discussed in previous work [45], [46]. In order to address these issues, Open RIZE encourages users to design interaction processes for social robots using a set of independent and reusable robot behaviors modeled as BTs. Therefore, modularity is not only a feature of the the software architecture, but it is also inherent to the organization of the generated robot behaviors. The approach put forth in Open RIZE takes advantage of the modular nature of BTs to organize and isolate robot behaviors (or sub-BTs) in a set of tables which users can easily update, delete, or reuse. Furthermore, Open RIZE users can decide whether to execute a specific robot action, a module, or a full sequence of robot behaviors. This degree of modularity is seldom available in block-based programming environments for Robotics, and constitutes *the first main novelty/contribution of the approach* presented and discussed in this article.

Almost all the state of the art VPL approaches make use of a server module coordinating all the other modules in the robot architecture, including those managing sensors and actuators, by using the *Request-Process-Reply* design pattern, also denoted as *Remote Procedure Calls* [47]. RIZE enables non-blocking and asynchronous communication between all the modules in the software architecture using the *Publish/Subscribe* design pattern [48], which enforces reusability, extensibility, maintainability, and robustness. Such an event-based approach in a VPL framework represents *the second main novelty* compared to previous work.

An important limiting factor characterizing the majority of programming tools for non-roboticists is their lack of accessibility for both developers and end-users [1]. On the one hand, many available solutions are made public as commercial software, and are not always available, or are difficult to integrate in real-world applications as components off-the-shelf (COTS) [1]. Moreover, these commercial EUD/EUP tools are generally targeted to specific robotics platforms. On the other hand, several open source projects assume the use of modules developed for a limited set of programming languages, or targeted at specific operating systems. Open RIZE is free for research purposes and enables the integration of modules developed in a wide variety of programming languages. It can also be executed on the most popular desktop and mobile operating systems. On a practical side, many EUD tools for Robotics require the use of the *command line* for installing main and third-party libraries, which is error-prone and subject to the help of experts. This issue affected also previous versions of RIZE [42], [45], which required the use of the Python Package Index (PyPI) using the command line [49]. In its current version, RIZE interfaces can be installed, uninstalled, and executed in either Windows or OSX using installation wizards. While most approaches in the literature are focused on programming tasks only, RIZE deals with all the aspects involved in the software development lifecycle by providing tools for an easy installation, configuration, re-design, debugging, scaling, and execution of applications for social robots. This approach represents *the third main contribution* of this article.

Involve potential users in the development of applications and tools is a keystone feature in software development. The RIZE framework and the Open RIZE interface has been designed and developed based on the feedback obtained from real end-users over several iterations involving the creation of HRI applications executed ''in the wild''. This variety of HRI contexts and real-world implementation we present in this article is rarely reported by developers of EUP/EUD tools for robotics.

III. THE RIZE FRAMEWORK

This section describes RIZE's architecture and presents an overview of the underlying concepts and available modules.

A. SYSTEM'S ARCHITECTURE

RIZE is an EUD framework for social robots built on top of a number of web-based technologies and able to exploit Component-based Software Engineering (CBSE) approaches [50]. Figure 2 shows the RIZE system's architecture. The architecture is multi-layered, and follows a conceptual *hardware to end-users* stacked pipeline.

In the lowest layer, i.e., the *Hardware Layer*, sensors, actuators, and whole robot platforms reside. These interact with low-level software modules (also referred to as *nodes*), which reside in the *OS Layer* and are in charge of exchanging various forms of data, and in general controlling hardware devices. These modules are executed by different operating systems. Currently, RIZE tools mostly focuses on Microsoft Windows and OSX. RIZE was also tested with success in different versions of Ubuntu. The data connection between these low-level nodes can be *local*, i.e., nodes run on the same workstation, *remote*, i.e., nodes are in different workstations connected via a certain networking structure, or their combination.

An essential linchpin used to help develop distributed applications is the *middleware*, which provides a set of

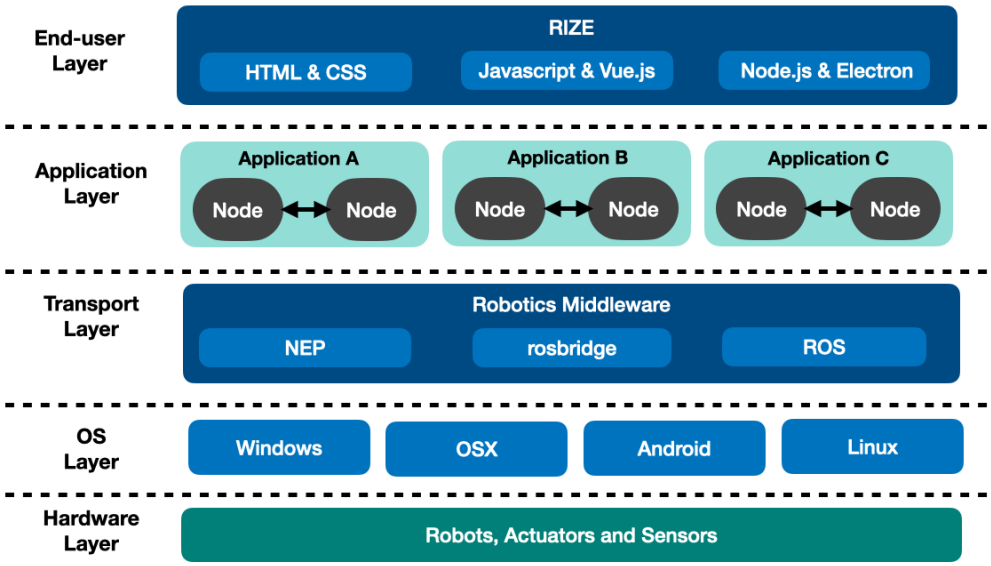


FIGURE 2. The general software architecture of a RIZE application. Low-level modules are developed on top of different robotics middleware. These modules are transparent to the end-user and are managed by graphical user interfaces using web-based technologies.

services and tools to enable communication between nodes. In the *Transport Layer*, RIZE supports software modules using the Robot Operating System (ROS) framework [51], and the rosbridge suite [52]. This support allows for an easy re-use of many open source and academic-oriented modules common in Social Robotics research. However, RIZE embeds also a middleware designed to support non-ROS software components, robots, and other computing devices. We refer to this communication framework as *Node Primitives* (NEP) [1]. NEP has been designed to allow developers who are not proficient in ROS/ROS2 development, or whose main focus is not Robotics-oriented, to easily integrate their modules.

Modules, which are part of the *Application Layer*, can be developed using different, modern programming languages and environments, such as C#, Python (2 and 3), Java, Matlab, and JavaScript (via Node.js). RIZE includes NEP in its installer package.

At the top layer of the architecture lies the RIZE programming environment, also referred to as the *End-user Layer*. A RIZE interface uses HTML/CSS for defining visual elements and JavaScript for the development and integration of web-based libraries, such as Google Blockly. It uses Vue.js for building user interfaces [53]. This JavaScript framework uses the *Model-View-ViewModel* (MVVM) methodology [54], which is a software architectural and structural pattern aimed at enforcing the separation and development of Graphical User Interfaces (GUIs). RIZE interfaces use Electron, i.e., a JavaScript framework for creating native applications with web technologies, and node.js, which is an event-driven and asynchronous JavaScript run-time environment used to create server-side applications. In this way, RIZE supports the creation of cross-platform desktop applications, which can be installed from user-friendly desktop installers. This approach avoids the use of command line instructions and the compilation of source code for software installation and execution. It also hides low-level tasks or interfaces, such as terminals, thereby enabling the development of production-ready programming environments applications for social robots. Moreover, web-based tools enforce an easy development of usable, modern, attractive, and beautiful interactive tools.

B. MODULES IN RIZE

A RIZE interface, such as Open RIZE, is composed of a *visual programming environment*, a set of *knowledge-making* and *action-making* modules, a *blackboard*, a *decision-making engine*, and a *master* module. We briefly explain the rationale behind these module types in the next few paragraphs, whereas more details are given in the Sections that follow.

A *visual programming environment* (VPE) is a graphical user interface that end-users can exploit to structure their programs. A VPE integrates at least one visual language, which can be based on rules, blocks, forms, or data-flows [13]. Open RIZE is an example of VPE based on the RIZE framework whereby users can define desired robot (social) behaviors. We discuss in greater details this VPE in Section IV. The main outputs of VPEs in RIZE are platform-agnostic and transport-independent representations of robot behaviors. For this, we use the JavaScript Object Notation (JSON) rather than a domain-specific or general-purpose programming language.

The *knowledge-making* and *action-making* modules represent input and output adapters that enable robots to sense and act in the environment, respectively. These modules can encode typical data collection, feature extraction and fusion, as well as control algorithms used in typical robot-based

architectures. They can also be programmed using different programming languages, and executed in different operating systems.

A *blackboard* is a behavioral design pattern used to support the storage and the dispatch of information to various modules, which allows them to interact [55]. As it is common in distributed architectures, communication among modules in RIZE is carried out via the exchange of messages, which are classified in topics. These messages are represented in RIZE as data structures in the blackboard module. Within RIZE, a *blackboard* module temporally saves and processes the output data of knowledge- and action-making modules. This information is also shared with other modules for various uses, including social behaviours and decision-making modules. Advanced blackboard approaches are designed to be independent from or easily adaptable to different decision-making strategies.

The RIZE *decision-making* module, or engine, is in charge of interpreting the behavior structure defined in the VPE, and managing the execution of robot behaviors accordingly. The module exploits the information made available by the *blackboard* module to guide robots to act according to the prescribed behaviors in the VPE, as well as the current environmental and contingent status. The module also publishes platform-independent action specifications for the control of robot's actuators. Then, a set of *action-making* module read to these messages and executes the robot behaviors. The *decision-making* engine can be designed using a pure AAI method, such as the one presented in [42], [45], or employing hybrid approaches, such as those discussed in Section V-C.

The NEP-based *master* module provides a discovery service for modules in a RIZE architecture instance, and thereby it manages the communication among software components using the NEP communication framework. The discovery service allows a given module to find, among available modules, those with a compatible required or provided interface in terms of topics. In this respect, a module can be considered as an independent computational process representing a basic element or building block of a robot-based application. Whilst different communication-related design patterns exist and have been discussed in the literature, RIZE adopts a simple publish-subscribe strategy for inter-RIZE module communication, with the aim of enforcing asynchronous data exchange and improving the overall robustness and fault-tolerance of the resulting software architecture. In order for a module to communicate with other modules in the architecture, a developer must specify its *required* (i.e., the topics it is subscribed to) and *provided* (i.e., the topics it is publishes information upon) interfaces. Every topic structure must be registered beforehand with the *master* node, which then manages the endpoint information for each topic. When a topic registration request is detected by the *master* module, an endpoint direction is assigned to a specific topic. This endpoint information is saved and sent to the modules that subscribed to that topic. Similarly, when RIZE modules must communicate with non-native RIZE modules, such as ROS-based modules, the discovery service is invoked to communicate with a ROS master node (in ROS 1.0) and the rosbridge server.

IV. OPEN RIZE PROGRAMMING INTERFACE

In this Section, we propose the Open RIZE interface as an initial prototype to prove the capabilities that the RIZE architecture and the associated tools can offer in adopting the EUD paradigm in real-world applications with social robots. The design aspects we considered are presented in the paragraphs below.

A. USER-CENTERED GUIDELINES

When working on Human-Computer Interaction (HCI) usability is an important aspect to enable a user to reach their goal using a system. Usability focuses on the creation of interactive products that are easy to learn, easy to remember, effective, efficient, and safe [56], [57]. To develop interfaces that provide high usability during interaction, researchers in HCI have proposed some design guidelines such as those listed in [58]. However, these guidelines always involve trade-offs, and their correct implementation depends on user activities, and the context of use [59]. Therefore, designers must juggle the constraints of the user, the context of use and the system in order to find a balance between task effectiveness and efficiency. In order to take into account the usability in Open RIZE and to achieve simplicity of use, we have attached importance to the following usability rules: consistency, management of feedback and recovery, and minimal design (Table 1). An interface that is consistent is an interface in which the way of using it is the same in the whole application. It means that the interface seems homogeneous in terms of visual elements (e.g., fonts, backgrounds, and colour), vocabulary, data format and positions of the different elements of data in the screen [58]. Taking consistency into account enable users to remember easily the procedures used in the application and then to use it more intuitively. To preserve consistency, designers can follow well-established design patterns or layouts. Examples are Google's material design [60] and Apple's Flat design [61]. By providing the appropriate feedback mechanisms or messages through the interface users can be informed of what is going on in the system. Therefore, for each action that the user performs through the interface, a clear and visible reaction should appear to enable the users to understand the system and maintain an interaction with it. An example of feedback implementation included in Open RIZE is to show visual elements that explain the state of the robot (e.g., when it is connected or disconnected to the network) as well as error messages and possible solutions. Programming a robot is in many cases an iterative trial-error process. Therefore, mechanisms enabling users to recover from both execution errors (e.g., create a new program that acts differently as expected) and unintended errors (e.g, deleting a behavioural block) must be implemented. Finally, a minimalist design seeks the reduction of unnecessary contents in term of information

TABLE 1. Usability guidelines adopted for the design of the rize interface.

Guideline	Description
Consistency	Use accepted standards or conventions to avoid confusions
Feedback	Clearly show the current state of the system and avoid mode errors (e.g., if the program is in execution or not)
Recovery	Errors must be easily recognized and recovered
Minimal design	Reduce information overload

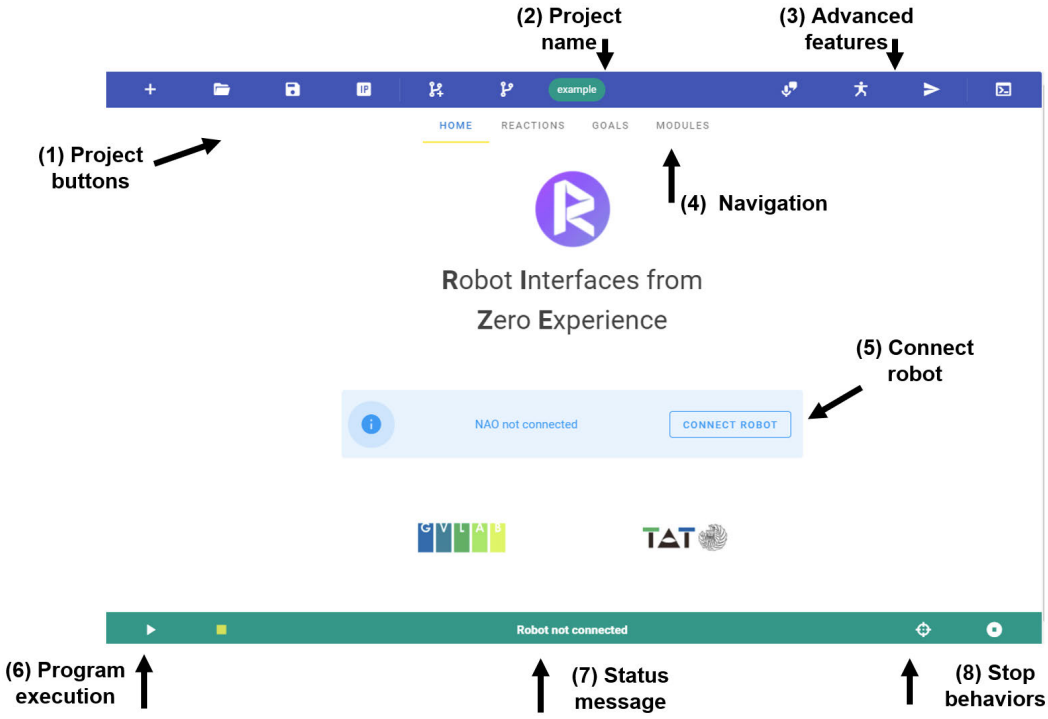


FIGURE 3. RIZE interface.

data, by only keeping those strictly necessary for the users to reach his tasks. It means that any "decorative" elements or not essential contents should be avoid or be made less visible. Examples of design strategies used in Open RIZE to keep its interface minimal are the use of generous whitespaces and limited colour schemes as well as the use of flat patterns and textures. The next section describes some of the most relevant mechanisms, visual elements, and tools implemented in Open RIZE for enabling the adoption of these usability rules.

B. DESIGN AND FEATURES OF THE OPEN RIZE INTERFACE

To design the graphical elements in the Open RIZE interface we use Vuetify.js, a material design framework [62]. As described in [60], material design is a group of "guidelines, components, and tools that support the best practices of user interface design". These guidelines were originally proposed by Google, and are used in many software products, examples being Gmail, Google Docs, Google Translator, Google Maps, and the Android and Google Search interfaces. These are, as a matter of fact, tools widely used by the generic user in everyday life. Therefore, the adoption of material design not only allows for the creation of user interfaces adhering to current aesthetic tendencies, but that are also more familiar and consistent. Figure 3 shows the home tab of the current version of Open RIZE. The interface is divided into 4 spaces with a toolbar on the top, a menu bar under the toolbar, the workspace, and a footer.

In order to reduce the user's cognitive load and preserve consistency, EUD tools typically use standard icons as well as tool-tips indicating the functionalities of each button. In the main interface, a set of buttons located in the top toolbar (in Figure 3 highlighted with the number 1) provides basic functionalities such as creating new projects, loading a project, saving the current project, setting a robot's IP address, creating a new version of the project, and loading a previous version of the project. The name of the project is located at the center of the top toolbar (indicated with number 2). On the right hand side of the top menu (number 3), a set of buttons is used to activate advanced features, namely activating the robot's speech recognition capabilities, defining new animations, sending speech commands for *Wizard of OZ* experiments, and opening the system's terminal for specific purposes. The menu bar consists of a set of tabs, collectively displayed in the Figure 3 with the number 4. This menu can be used to switch among different Open RIZE's workspaces. Each workspace consists of a home page

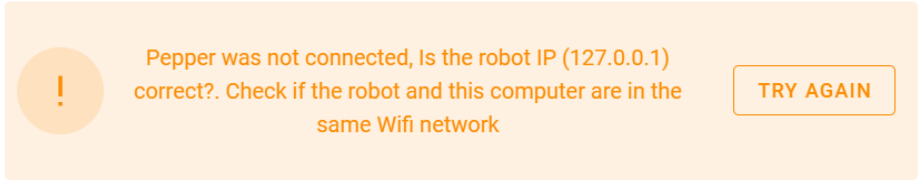


FIGURE 4. An alert message displayed when the robot is not connected.

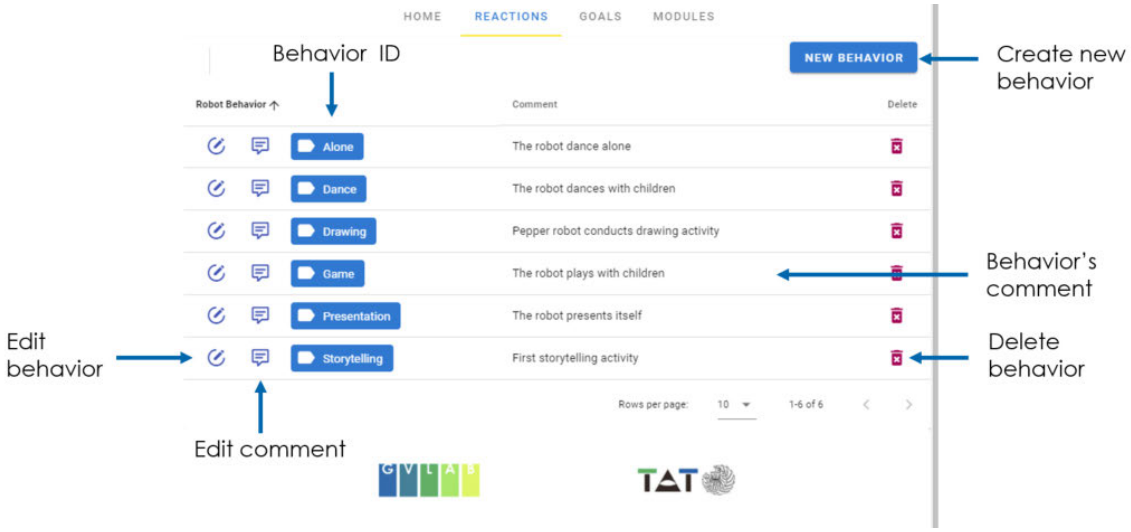


FIGURE 5. Organization of robot social behaviors in different modules.

(shown in Figure 3) and a set of tables where users can add, delete, or update robot behaviors. The connection with robots is established by pressing the button represented with the number 5 in the Figure 3. This button is inside a graphical element known as ''alert'', where the connection status of the robot is displayed. This element can display different colors and instructions depending on the connection status. An example is shown in Figure 4, whereby a warning message is displayed when Open RIZE is not able to connect to the robot. A footer menu is used to start and stop an Open RIZE program, as indicated by the number 6, as well as to show the robot status (number 7). Finally, the button represented using the number 8 is used to stop interactive events or behaviors, such as speech recognition and face tracking.

One of the most relevant usability issues in data-flow and block-based programming environments is the need to deal with very large programs requiring the use of many graphical elements [13]. Specifically, the more a program is large and complex, the less readable it becomes and the more difficult it is to find a particular piece of code. In order to overcome this problem, robot social behaviors in Open RIZE are modeled as independent sub-programs and are organized in lists. Each robot behavior (or item) in a list is associated with a unique identifier, and can be provided with a comment, which can be used by end-users to include explanations about each behavior's functionalities. An example is shown in Figure 5. Robot behaviors are classified as *reactions* and *goals*, which are explained in depth in Section V. Furthermore, Open RIZE provides graphical elements encapsulating a sequence of robot actions in *modules*. Open RIZE includes a predefined set of lists of sub-programs, whereby the user can add, delete or update relevant behavioral modules. Each sub-program is modeled as an independent BTs. Therefore, a Open RIZE project consist of a set of modular behaviors or sub-BTs organized by categories in lists. When selecting an item from one of these lists, a new interface appears. An example is shown in Figure 6.

Programming interfaces in Open RIZE provide error prevention and recovery mechanisms, such as confirmation options before performing risky actions, undo and redo buttons. Moreover, Open RIZE features a basic versioning control approach enabling users to recover a desired version of the selected sub-programs. The menu at the bottom of the interface in Figure 6 is used to execute the current module as well as to inspect the code generated by the current program. This code for social behaviors is represented as BTs.

C. COMBINING BLOCKS AND FORMS

Open RIZE is a novel approach to program complex robot social behaviors, expanding the capabilities of such blocks-based programming tools as Google Blockly with form-filling interfaces. In our use cases, blocks are used to

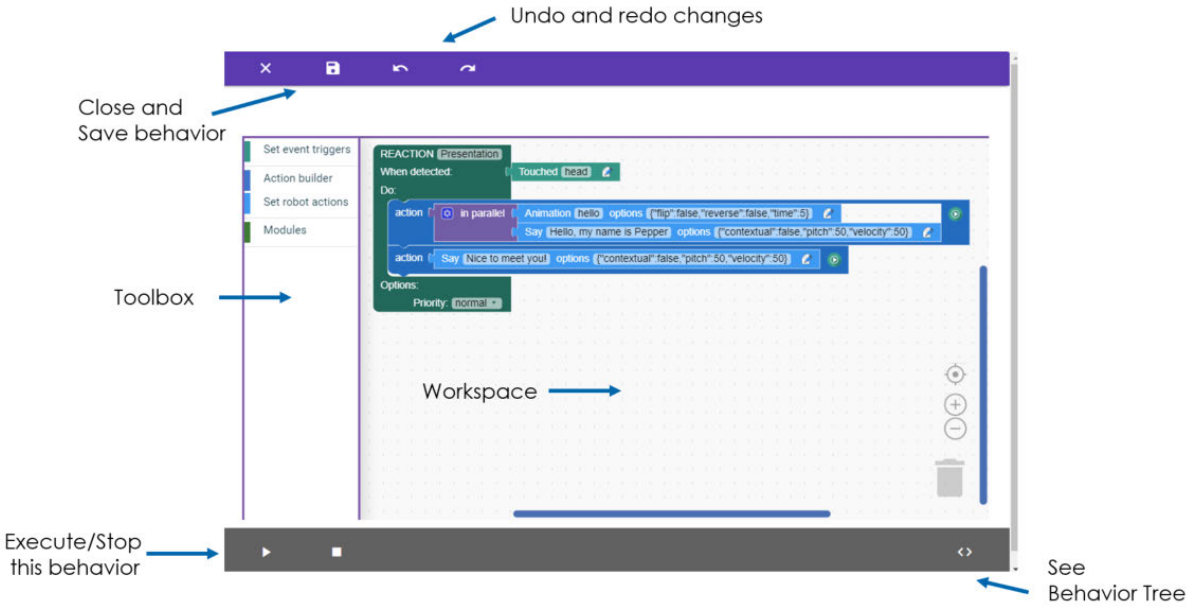
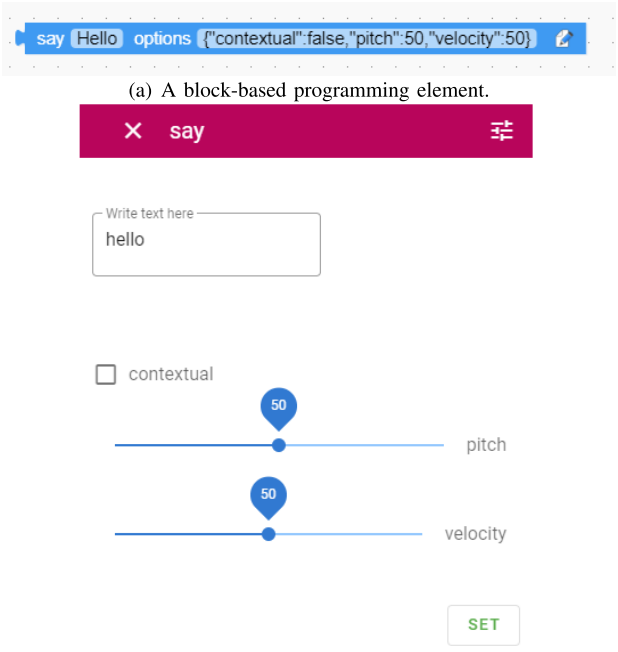


FIGURE 6. Example of a programming environment where end-users can define independent robot social behaviors.



(a) A block-based programming element.

(b) A form interface used for parameters tuning.

FIGURE 7. Example of a block representing a robot action an its form interface.

organize and structure robot behaviors as a combination of social skills, modules to manage perceptual inputs and to perform robot actions, whereas the associated form-filling interfaces are used to tune their parameters.

Figure 7 shows an example of a block element representing a given social skill, and its related form-filling interface, whereby robot text-to-speech capabilities can be modified. In this case, the user can alter either the velocity or the pitch (or both) of the robot's voice, the text to utter, and select whether this action requires the concurrent execution of any contextual animation. Available form elements include sliders for numerical inputs, text boxes for string-based inputs, check boxes for Boolean inputs and drop-down lists for the selection of elements in a given library, e.g., a list of emotional expressions, sounds or animations that the robot may make available.

D. MODIFYING ROBOT SOCIAL BEHAVIORS IN OPEN RIZE

Any visual element in Open RIZE can be modified using a Python tool called Open RIZE Blockly Generator. As shown in Figure 8, developers can add new robot social behaviors to the Open RIZE programming environment by defining their main attributes in a JSON file. These specifications are read by a Python tool that generates the JavaScript and HTML code defining the design of block elements and their parameters in the form-filling interfaces of each behavior. All the guidelines for adding new robot social behaviors to Open RIZE are described in the documentation web page of the Open RIZE Blockly Generator tool [63].

As an example, let us suppose speech capabilities must be added, as shown in Figure 7. If a speech-related behavior is available (maybe it corresponds to a set of specifically developed ROS nodes), the associated parameters can be defined using a formalism similar to that reported in Listing 1. In JSON notation, a developer can define an intuitive title of for the behavioral *primitive*, which will be therefore visualized as a block element (line 2). In this case, the speech-related behavior is denoted as an imperative action, i.e., *say*. Then, it is necessary to specify the behavior's *type*, i.e., whether the primitive is related to robot's perception (an input) or action (output) capabilities (line 3). A *description* of the block's functionalities can be included in the form

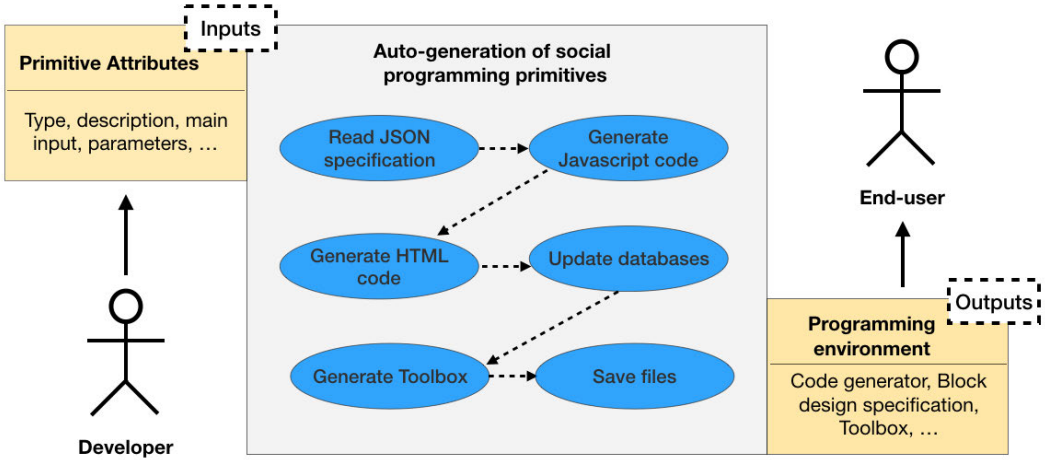


FIGURE 8. Schematic overview of the processes involved in the generation of new functional requirements for in Open RIZE environment.

```
1  {
2      "primitive": "say",
3      "type": "output",
4      "description": "The robot says something",
5      "input": "string",
6      "options": {
7          "contextual": "bool",
8          "pitch": "percentage",
9          "velocity": "percentage",
10     }
11 }
```

Listing. 1. Example of a social behaviour definition in Open RIZE.

of a text string (line 4). This description is shown when an end-user selects the block. The *input* key is used to specify the main parameter associated with the block, in this case a text string indicating the sentences that will be uttered by the robot. A set of optional inputs can be specified using the *options* key. Each JSON description of the social behaviors is parsed by a Python script. This script generates the respective HTML and JavaScript code that defines the visual elements of the block-based programming environment and form-filling dialogues (used for change the parameters of each behavior block) in Open RIZE.

V. THE OPEN RIZE's COGNITIVE ARCHITECTURE

Figure 9 shows the scheme of an interaction-oriented cognitive architecture implemented in Open RIZE and adopted in a series of real-world applications and software development workshops with non technically-skilled end-users. The architecture is composed of a set of knowledge-making modules (i.e., sensory and perception behaviors), a blackboard system, a hybrid decision-making engine, and a set of robot action-making modules. Communication among modules is mainly managed by the NEP master node. Specific applications using nodes developed in ROS require the integration with a ROS master node and the rosbridge server.

A. KNOWLEDGE-MAKING MODULES

Information that a robot can collect from the environment is intrinsically multi-modal and heterogeneous in nature. Therefore, it can be obtained via different sensory modalities. Open RIZE assumes the use of microphones, cameras, touch and range sensors.

From a purely software architecture perspective, each sensor is managed via a specific *Sensor-Device* design pattern, i.e., low-level sensor device drivers are encapsulated in software modules that can exchange messages via a publish-subscribe mechanism. Overall, these modules are part of the Open RIZE's *Sensory system*. Apart from relevant filtering algorithms and data preparation when necessary, such a system provides low-level, raw information to the *Perceptual system*, where data is structured and analyzed using a series of social behaviors.

The Perceptual system consists of a number of knowledge-making modules, which include social behaviors aimed at making conceptualized data available to the blackboard for further analysis and use by the decision-making engine. Typical robot behaviors reside in this system, including state of the art algorithms for speech recognition, emotion recognition, face/object recognition, and the interpretation of touch and contact phenomena, as well as the interpretation of human proxemics behaviors. The internal organization of the Perceptual system is not limited to a number of sensory data streams, insofar as data must be combined in complex structure to be aggregated. Overall, the modules in the Perceptual system generate output data that can be interpreted as social behavioral primitives [26]. These data are encoded using JSON serialization and dispatched to the blackboard, which stores them for later processing.

B. THE BLACKBOARD MODULE

In Open RIZE, knowledge-making modules, the blackboard, and the decision-making engine are separated modules that communicate with each other using a message-driven scheme. All the modules in the Perceptual system write high-level data on the blackboard. The format of these data must be defined in advance using appropriate data structures, which are implicit in the modules' provided interfaces. A suitable technique to structure data to be transmitted via

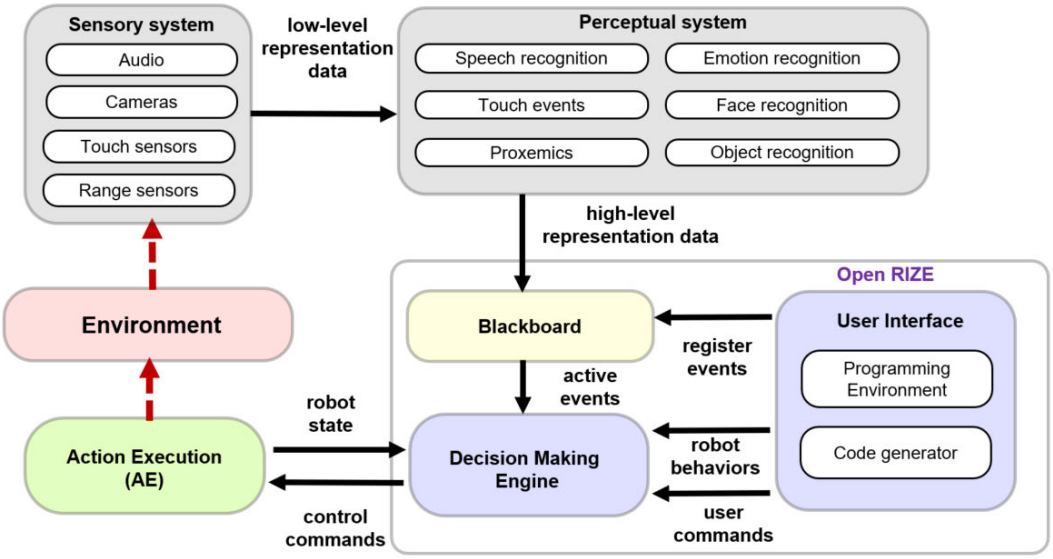


FIGURE 9. Open RIZE's software architecture.

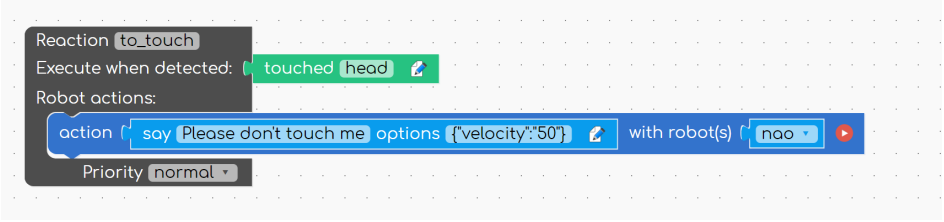


FIGURE 10. Example of a simple reaction behavior.

publish-subscribe is the use of *key-value* pairs. While a *key* is a label associated with a specific datum, a *value* can be any suitable data type, including integer, float, double, string, list, or dictionary, among others. In order to ground these structures within well-formed messages, we decided to use JSON, which is an open-standard, programming-language-agnostic and transport-independent format used to transmit and save data objects represented with *key-value* pairs.

In a broad sense, a blackboard can be used to store information with varying levels of abstraction. By itself the blackboard is agnostic with respect to the stored information. However, a general recommendation to design AAI modeling tools for non-technically skilled end-users is to adopt a high-level data representation, which is as close as possible to the domain knowledge end-users possess to support its reasoned use [26].

C. THE DECISION-MAKING ENGINE

In RIZE, decision making (i.e., behavior selection and sequencing) is managed by BTs. Although BTs outperform FSMs in terms of a better trade-off between modularity and reactivity, they are also characterized by a number of drawbacks. One of them is their computational cost. A classical decision-making engine only based in BTs evaluates BTs specifications from the root node until a node with a status of *success*, *failure* or *running* is reached. This approach was used in an experimental version of RIZE [45]. However, each new iteration requires to fully re-evaluate the tree, checking all preconditions and states of the executed actions. This approach enables reactivity in BT but also produces a large number of unnecessary checking conditions (call from the blackboard) which can produce prohibitive computational cost for large BTs. An option to solve this problem has been denoted as BT with memory nodes [38]. In this approach, rather than examining the entire tree structure, execution is resumed from some check-points avoiding those nodes that had previously returned *success* or *failure*. However, these compromise the reactivity features of the system, because the robot can react only after a *success* or *failure* status is returned for the full tree.

We deal with the problem of excessive checking of conditions associated with the classical execution of BT by proposing a novel decision-making engine composed of two levels. At the higher level, a mechanism based on FSMs and priority selection manages the overall execution of robot behaviors with strict and simple control of the high-level robot decision-making. At the low-level, the system executes the user-defined modules. These modules or sub-BT are classified in two main types: *reactions* and *goals*.

Reactions. These modules are small sub-BT which can be activated after the detection of a particular social stimulus. An example modeled using Open RIZE is shown in Figure 10. In this example, the action of text to speech is defined to be executed every time the robot is touched on the head

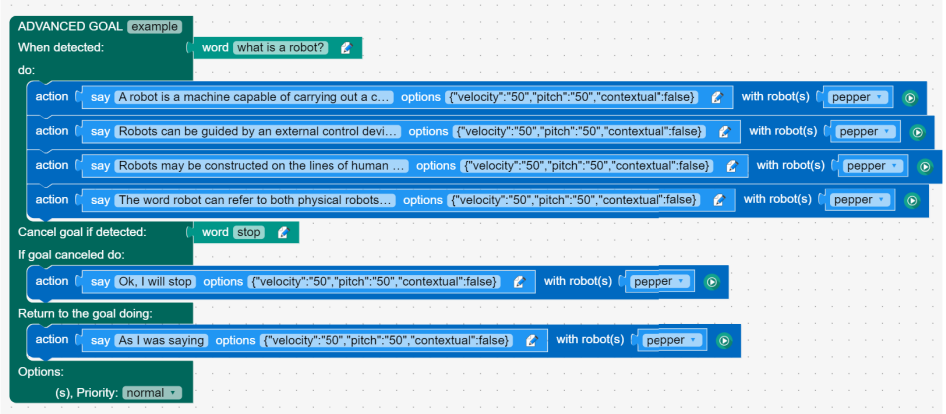


FIGURE 11. Example of a simple goal behaviour.

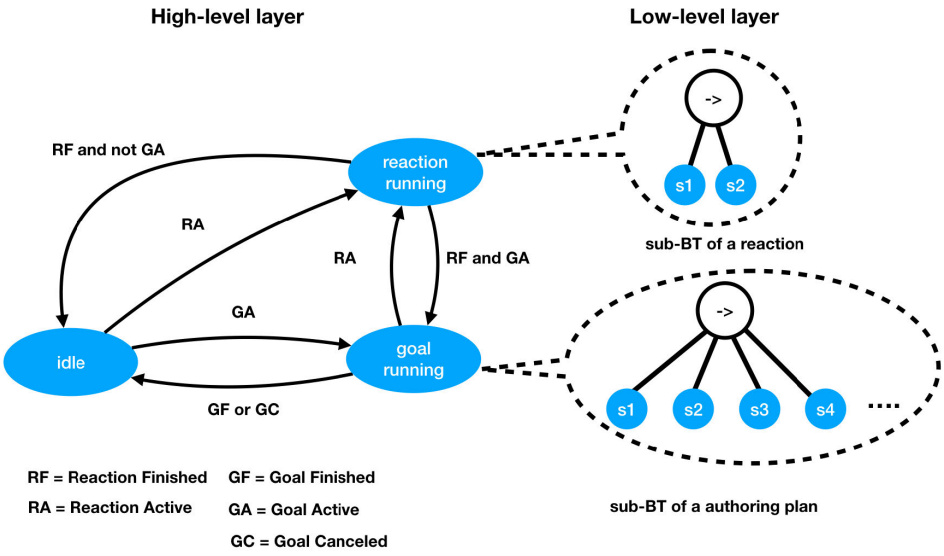


FIGURE 12. Simplified diagram representing the proposed hybrid FMS and BT decision-making approach.

(the trigger condition). The proposed approach not only facilitates the creation of Trigger-Action Programming like [64] (if-then) behaviors but also enables to define the priority of these behaviors. This priority value can be defined by the user as *low*, *normal* and *high*.

Goals or authoring plans are defined as sequences of actions, or other complex behaviors. A goal is activated or canceled if some sets of conditions are met. Unlike reactions, goals can continue execution after their preemption by some reaction with higher priority. In the proposed approach only one goal or one reaction can be managed at the same time. A simple example of a goal behavior is shown in Figure 11. In this example, the goal is activated when the speech recognition system detects the phrase "*what is a robot?*". Then, the robot executes the main behavior (a set of 4 actions composed of say social primitives) until all the actions are completed or until the condition to cancel this behavior is met. In this example, when the behavior is canceled the robot will say "*Ok, I will stop*". Moreover, if this behavior is preempted from completion to attend a high priority reaction (for example, head touched), at the end of the reaction, the flow of the program can return back to the goal and continue with its execution. Furthermore, before executing the list of goal's actions that were not yet completed, the robot can execute some action to notify humans that it will continue with the original goal. An example is shown in Figure 11. The return behavior is modeled as a say action with the text "*As I was saying*".

The approach used to execute reactions and goals is similar to the BT with memory method [38]. However, the proposed approach uses the FSM defined in the high-level layer of the decision-making engine shown in Figure 12 to enable reactivity. For this, a register of the conditions that activate or cancel the defined reactions or goals, and their priorities is first done. A program in Open RIZE maintains its execution in the *idle* state until a set of social primitives that activates

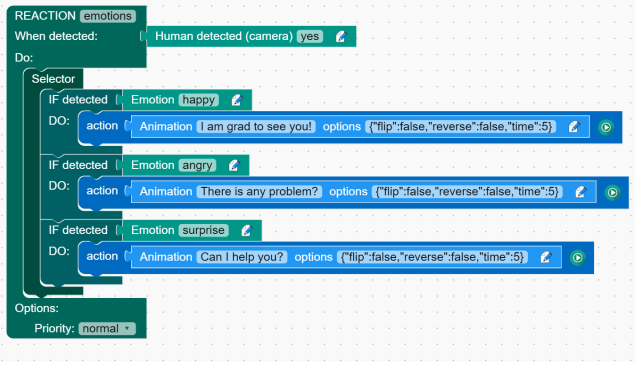


FIGURE 13. A simple example of how selector can be used to change the robot's behavior based on the current state of the interaction environment.

some goal or reaction is detected in the blackboard. From Figure 12, if a reaction is activated (RA is true), the state of the FSM changes to *reaction running*. Then, the sub-BT which corresponds to the active reaction is executed until its tree returns *success* or *failure*. The execution of this reaction can be preempted if another reaction with higher priority is triggered.

When a goal is active (GA is true), the FSM state changes to the *goal running* state. If a goal is preempted by a reaction before finishing the execution of all its actions (RA is true when GA is true), this goal will resume its execution from the last performed action once the reaction that caused the preemption has been completed (RF is true and GA is true). If the current goal completes its execution (GF is true) or the goal is canceled (GC is true) the interaction state changes back to *idle*. Goals can also resume their execution or cancelled at any time, this is done by an an FSM located inside the goal running state.

Other types of behaviors that can be part of reactions or goals are *sequences* and *selectors*. *Sequences* encapsulate a list of actions, selectors, or other sequences and thus enable the easy re-use of parts of the code in different modules in the programs. A sequence can be executed in two different ways: (i) executing all the elements in the sequence in order; or (ii) selecting only one element randomly for its execution. These behaviors can be created in the *modules* list as independent behaviors. *Selectors* enable the robot to behave based on the current state of the environment. Figure 13 shows a simple example of how a robot can react in different ways depending on the detected human emotion.

D. ACTION-MAKING

The primary function of this module is to manage the execution of the robot commands sent by the decision-making engine. It is important to highlight that to enable a BT-based architecture to react to changes, periodical feedback of the state of the actions must be done between the *decision-making engine* and the *action-making* [38]. For this, the *action-making* publishes the current state of the robot's actions back to the *decision-making engine*. This information is used for the *decision-making engine* to manage the robot behaviors. The implemented *action-making* modules in Open RIZE are written in Python and can also be connected to low-level control modules (e.g. Moveit [65] for motion planning and execution) in other programming languages using NEP or ROS frameworks. To help with the integration and re-use of robot functionalities written in Python, we provide the *rize* package. This Python package can be installed using *pip* and implements a simple mechanism based on FSMs that manages the execution of the functions executing the robot behaviors. The outputs of modules using the *rize* Python package are messages with the current status (success, running or failure) of the action being executed.

VI. VALIDATION WITH END-USERS AND IN THE WILD SCENARIOS

The creation of user interfaces generally is a user-centered iterative process that required the inclusion of end-users in the analysis, design, evaluation, and implementation phases [66]. However, in the robotics community, the inclusion of end-users in the development of HRI scenarios as well as their implementation in real scenarios is still rare [3], [67]. Instead, the majority of existing HRI studies for robotics are performed in laboratories or other semi-structured scenarios. Moreover, the evaluation of EUP/EUD interfaces and HRI scenarios is often performed with convenient samples (i.e., laboratory colleagues or engineering students), instead of users representative of the target audience addressed [67].

The target users of Open RIZE are non technical persons who need to create scenarios of Human-Robot interaction. Through collaborative project with various people in this situation, we refined incrementally Open RIZE. End-users were invited to directly program and execute the desired robot behaviors using several versions of the Open RIZE interface. Applications developed by end-users include robots on stage (performed in 2017 and presented in [42]), robots in domestic settings (performed in 2018 and presented in [68]), Children-Robot Interaction (CHI) (performed in 2019 and presented in [46]), and co-design workshops (performed in 2020 and which preliminary results are presented in [69], [70]). For each collaboration, the main outcomes were: (i) the integration of novel methods and technologies to satisfy the requirements of end-users, (ii) design of a block-based programming environment adapted to the context of the end-user applications, and (iii) evaluation and implementation of applications developed by end-users in real and public settings. We briefly describe main objectives of these collaborative projects below. Main outcomes, requirements, and technologies implemented in each of these projects are summarized in table 2.

The initial version of RIZE used a programming interface where robot behaviors are modeled as simple IF-THEN-ELSE rules which launched a set of sequential behaviors. As described in [45], this interface was developed on top of a Python-based web server and basic functionalities of

TABLE 2. Main detected issues and implemented technologies for each iteration with real end users.

Application area	Main user needs	Main detected issues	Implemented technologies
Theatres and museums [42]	An easy-to-use programming environment	Rule-based system limits the complexity of the robot behaviors that end users can design	NEP, Flask, Google Blockly, Rule-based systems
Domestic environments [68]	Creation of reactive and goal-oriented behaviors	The use of Python to back-end RIZE programming interfaces produce error-prone installation steps	Behavior Trees (BTs), Blackboard systems
Children-Robot Interaction [46]	Creation of semi-autonomous behaviors	Block-based programming environments present re-usability issues	Node.js, Vue.js
Co-design workshops [70]	Use of a robot simulator, integration with ROS-based robots, version control	Bugs in some functionalities	Vuetify, electron, form-filling interfaces, ROS, rosbridge

Google Blockly. We use this initial version to enable comedians and social researchers to design and develop simple interactive scenarios with social robots [42]. In this way, professionals were able to explore new paradigms supporting their profession. An example of application shown in [71] is the design of a set of short comical sketches between human actors and a NAO social robot. For the creation of interactive scenarios in domestic environments [68] and CHI [46], it was required to increase the reactivity, re-usability, and complexity of those behaviors designed by end-users. In these two cases, social robots required to dynamically react to multi-modal inputs and perform different types of activities, such as games, exercise routines, dance, display current weather conditions, among others. To enable the easy creation of more reactive, complex, and dynamic behaviors, we substituted the rule-based systems as the main programming approach by Behavior Trees. We also substituted the proposed Python-based web-server developed in [45] by a Node.js to back-end RIZE interfaces (as described in section III). In comparison with server-side web frameworks in Python, Node.js is faster, scalable, event-based, and asynchronous. Moreover, applications developed with Node.js can be easily installed in most modern desktop-oriented operating systems without the need to install additional third-party libraries. These new versions of Open RIZE enable social researchers to design long-term HRI scenarios in 5 Japanese and 5 French home environments [68] and several short-term activities in kindergartens with humanoid robots [1]. In these applications using NAO and Pepper robots, we implemented the NAO software development kit version 2.4.3 in Python for the development of the basic software modules.

In 2020, a collaboration was set up with researchers in Human-Robot Interaction from Monash University with the aim of better understanding how the general end-users understand robots overall and the possible roles robots could take in public spaces. This collaboration aimed to use Open Rize in several workshop sessions where participants had to create behaviours to put a robot into action. These workshops were organized and managed by researchers at Monash University, Australia, as part of the RiPS project [70]. Unlike previous applications using Open RIZE and due to the limitations of the COVID-19 lockdown, all these workshops were conducted via a video conferencing platform (Zoom). For these workshops, staff members of the Monash University with no prior experience with robots or programming were recruited [70]. These staff members belong to faculties of architecture, arts, and criminology. The same 11 participants realised the first two workshops, and 11 other participants integrated the third workshop.

The first two workshops were a two-hour session divided into three steps. The first step aimed at collecting data through questionnaires on users' general impression of robots and their opinion on robots in public spaces. The second step consisted of having participants discuss in groups of four about behaviours of a robot for public spaces. This step was a participatory step enabling participants to co-design behaviours of a robot. A brief demonstration of Open RIZE was done at the beginning of the step be fore the discussion starts. When exchanging together, participants indicated to a researcher in the lab which actions to realise on the interface of Open RIZE to create behaviours they wanted the robot to perform. In the first workshop, participants could see results of their actions through a simulator of the Pepper humanoid robot. In the second workshop, a real Pepper robot in the lab executed the programmed behaviors that participants could see remotely. The goal of this step was to deepen their understanding of robots and to elicit concrete discussions on robotics technology. In the last step, participants discussed together regarding suitable scenarios and robots' behaviours for public spaces. A more detailed explanation of the experimental protocol and results of the first and second workshops are reported in [69], [70], [72].

Following the previous workshops, a third workshop was set up with similar goals. The protocol of this workshop consisted of four steps. The first three steps were the same as the three steps of previous protocols. Figure 14 shows an example of a simple program developed by participants in the third workshop. In the fourth step realised after the session, participants were asked to fill in form online questionnaires regarding robot self-efficacy perception, perceptions of robots, workshop design, workshop tools and expectations of future robots' use. One of this questionnaire was the System Usability Scale (SUS), giving an idea of the level of usability of Open RIZE. SUS consists of 10 Likert items with five response options varying from strongly agree (5) to strongly disagree (1). This questionnaire is an industrial standard that researchers and practitioners should strongly consider using [73]. SUS allows a reliable and valid evaluation of

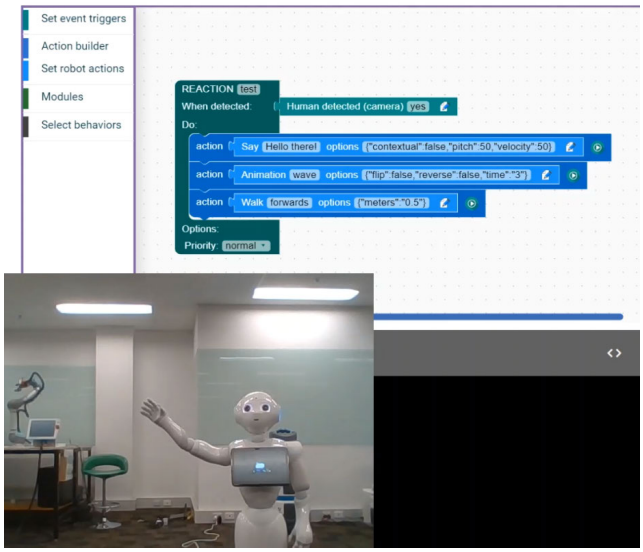


FIGURE 14. Example of a program developed by participants of the third workshop.

a wide variety of products and services such as hardware, software, websites, and applications. Moreover, it can be used in a small group of people and still offer reliable results with twelve persons [74]. To calculate SUS score is required to follow the next steps: (1) subtract 1 from the score obtained in odd questions, (2) for each even question subtract their value from 5, (3) add the values obtained from step 1 and 2 to the total score, (4) multiply the total score by 2.5 [73], [75]. SUS scores are between 0 and 100. Interfaces above 68 are considered to have above average and acceptable usability [75]. It is important to highlight that SUS scores are not percentages [75]. However, they can be interpreted as percentile values. For example, a SUS value above 80.3 is considered to be an excellent score that less than 10 % of interfaces have. Considering Open RIZE, the final SUS score is 72.73, which represents an acceptable usability score [73], [76]. Table 3 shows the mean and standard deviation (σ) for each SUS question. From the 11 participants in the third workshop, only 10 answered the SUS questionnaire. Aspects specially well-valued by the workshop participants were the easy-to-use and easy to learn aspects of Open RIZE (Questions 2,3,9, and 10). Participants also felt that some functionalities could be improved (question 5). The SUS score is a first evaluation of usability of Open RIZE, and possible biases have to be taken into consideration, in particularly, participants did no directly used Open RIZE and they used it in a group, implying that perception of usability could be different from an individual and direct use. Even if this result has some biases, we consider that this scale gives an initial usability evaluation that should be further developed through user testing.

In addition to the overall perception of the usability of Open RIZE through the SUS, this third workshop enabled to get some feedback regarding the functions of Open RIZE during the open discussions at the end of the session, and through the final questionnaires. Some of the most relevant functions suggested by participants are to allow communication with other electronic devices (e.g., smartphones and smartwatches) and more options enabling remote control of robots. The addition of the two first functions is a trivial task thanks to the modular nature of the proposed software architecture. This integration can be performed by using mobile applications built on top of NEP or the rosbridge suite, which can also enable the streaming of data between desktop PC and Android or iOS devices. Participants suggested also other desired functions especially to be able to manage the Pepper's tablet. This will guide the next iterations in the development of novel programming interfaces based on the RIZE software architecture and tools. A future session of this set of co-design workshops will include the use of the Fetch service robot.

TABLE 3. SUS questions and results.

N	Question	mean	σ
1	I think that I would like to use this system frequently	3.7	1.10
2	I found the system unnecessarily complex	1.5	0.50
3	I thought the system was easy to use	4.4	0.66
4	I think that I would need the support of a technical person to be able to use this system	2.6	1.01
5	I found the various functions in this system were well integrated	3.6	0.91
6	I thought there was too much inconsistency in this system	2.0	0.63
7	I would imagine that most people would learn to use this system very quickly	3.9	0.83
8	I found the system very cumbersome to use	2.0	0.77
9	I felt very confident using the system	3.7	1.00
10	I needed to learn a lot of things before I could get going with this system	2.1	0.83

VII. CONCLUSION AND FUTURE WORK

This article presented the software architecture and main software tools of the RIZE framework. As shown in this article, RIZE was designed to be a modular, distributed and accessible framework for building robot programming interfaces. The approaches and tools used in the development of RIZE enabled the implementation of different HRI applications in open, public and social settings. Moreover, the software architecture and modular design of RIZE allowed for an easy adaptation to the dynamic needs of users. This level of flexibility is rarely presented in EUD/EUP solutions for social and service robotics, which are often context, robot, programming language, or operating system dependent. Furthermore, the use of distributed and component-based approaches such as NEP and ROS enable the easy extension to other HRI areas by the re-use of open-source and academic software. Therefore, future work will focus on exploring novel contexts, such as educational and industrial scenarios. RIZE can also be complemented with a set of pre-built modules for knowledge acquisition using state-of-art deep learning methods, advanced emotional expression, and learning from demonstration. We also plan to explore novel and hybrid approaches between different types of visual languages and decision-making approaches for enabling the easy prototyping of more complex and dynamic robot behaviors. Rather

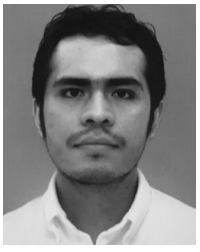
than propose a static or monolithic programming environment, such as most previous works, RIZE is designed as a modular and adaptable EUD research framework for robotics, which can be easily interfaced with code from other HRI projects and new robotics platforms.

Open RIZE is an example of a programming environment built on top of the RIZE software architecture and tools. This interface was designed to enable the easy and rapid development of common HRI behaviors. The technological suitability of Open RIZE was proved in several applications with different types of real end-users. The initial usability evaluation using SUS indicated that Open RIZE seems to be easy to use and learn. However, unlike other applications performed using a RIZE-based interface and due to the limitations of the COVID-19 lockdown, participants of performed co-design workshops were not able to directly interact with the interface and robots. This limitation can bias the obtained results of the SUS questionnaire. Therefore, more usability studies will be needed. A possible short-term solution to deal with the current issues due to COVID-19, which hinders the performance of HRI and HCI experimental evaluations, is enabling the remote programming and control of robots, which can be performed using communication tools such as rosbridge. Due that RIZE was designed to be independent of the VLP implemented (i.e., programming interfaces in RIZE can be based on blocks, forms, data-flows, or a mixture of different types), comparisons between different types of VPL approaches will be performed. For evaluating these different types of VPL, it will be required to consider objective (e.g., the time users required to complete a specific task or the complexity of the programmed behaviors) and subjective metrics (e.g., usability). This brings us to consider realizing user-centered research and testing that guide the development of novel programming interfaces, similar to Open RIZE, that not only can be adapted to different robotics platforms and HRI projects but also be able to deeply meet user's expectations.

REFERENCES

[1] E. Coronado and G. Venture, "Towards IoT-aided human–robot interaction using NEP and ROS: A platform-independent, accessible and distributed approach," *Sensors*, vol. 20, no. 5, p. 1500, Mar. 2020.
[2] G. Fischer, "End user development and meta-design: Foundations for cultures of participation," in *End-User Computing, Development, and Software Engineering: New Challenges*. Hershey, PA, USA: IGI Global, 2012, pp. 202–226.
[3] M. Jung and P. Hinds, "Robots in the wild: A time for more robust theories of human-robot interaction," *ACM Trans. Hum.-Robot Interact.*, vol. 7, no. 1, pp. 1–5, May 2018.
[4] S. Sabanovic, M. P. Michalowski, and R. Simmons, "Robots in the wild: Observing human-robot social interaction outside the lab," in *Proc. 9th IEEE Int. Workshop Adv. Motion Control*, Mar. 2006, pp. 596–601.
[5] D. Glas, S. Satake, T. Kanda, and N. Hagita, "An interaction design framework for social robots," in *Robotics: Science and Systems*, vol. 7. 2012, p. 89.
[6] T. Kanda and H. Ishiguro, *Human-Robot Interaction in Social Robotics*. Boca Raton, FL, USA: CRC Press, 2016.
[7] F. E. Ritter, G. D. Baxter, and E. F. Churchill, "User-centered systems design: A brief history," in *Foundations for Designing User-Centered Systems*. London, U.K.: Springer, 2014, pp. 33–54.
[8] N. B. Hansen, C. Dindler, K. Halskov, O. S. Iversen, C. Bossen, D. A. Basballe, and B. Schouten, "How participatory design works: Mechanisms and effects," in *Proc. 31st Austral. Conf. Hum.-Comput.-Interact.*, Dec. 2019, pp. 30–41.
[9] G. Fischer, "End-user development: From creating technologies to transforming cultures," in *Proc. Int. Symp. End User Develop.* Berlin, Germany: Springer, 2013, pp. 217–222.
[10] F. Paternò and V. Wulf, *New Perspectives in End-User Development*. Cham, Switzerland: Springer, 2017.
[11] B. R. Barricelli, F. Cassano, D. Fogli, and A. Piccinno, "End-user development, end-user programming and end-user software engineering: A systematic mapping study," *J. Syst. Softw.*, vol. 149, pp. 101–137, Mar. 2019.
[12] Enrique Coronado. *Open Rize*. Accessed: Jun. 1, 2020. [Online]. Available: https://github.com/enriquecoronadozu/Open-RIZE-beta
[13] E. Coronado, F. Mastrogiovanni, B. Indurkhya, and G. Venture, "Visual programming environments for end-user development of intelligent and social robots, a systematic review," *J. Comput. Lang.*, vol. 58, Jun. 2020, Art. no. 100970.
[14] P. Bachiller-Burgos, I. Barbecho, L. V. Calderita, P. Bustos, and L. J. Manso, "LearnBlock: A robot-agnostic educational programming tool," *IEEE Access*, vol. 8, pp. 30012–30026, 2020.
[15] B. Jost, M. Ketterl, R. Budde, and T. Leimbach, "Graphical programming environments for educational robots: Open Roberta–Yet another one?" in *Proc. IEEE Int. Symp. Multimedia*, Dec. 2014, pp. 381–386.
[16] N. Park, "Application and analysis of steam using education programming language in elementary school," *Int. Inf. Inst. Inf.*, vol. 16, no. 10, p. 7311, 2013.
[17] M. Á. Conde, C. Fernández, J. Alves, M.-J. Ramos, S. Celis-Tena, J. Gonçalves, J. Lima, D. Reimann, I. Jormanainen, and F. J. G. Peñalvo, "RoboSTEAM—A challenge based learning approach for integrating STEAM and develop computational thinking," in *Proc. 7th Int. Conf. Technol. Ecosyst. Enhancing Multiculturality*, Oct. 2019, pp. 24–30.
[18] C. Paxton, A. Hundt, F. Jonathan, K. Guerin, and G. D. Hager, "CoSTAR: Instructing collaborative robots with behavior trees and vision," in *Proc. IEEE Int. Conf. Robot. Automat. (ICRA)*, May 2017, pp. 564–571.
[19] F. Steinmetz, A. Wollschläger, and R. Weitschat, "Razer—A HRI for visual task-level programming and intuitive skill parameterization," *IEEE Robot. Autom. Lett.*, vol. 3, no. 3, pp. 1362–1369, Jul. 2018.
[20] A. Carfì, J. Villalobos, E. Coronado, B. Bruno, and F. Mastrogiovanni, "Can human-inspired learning behaviour facilitate human–robot interaction?" *Int. J. Social Robot.*, vol. 12, no. 1, pp. 173–186, Jan. 2020.
[21] V. Villani, F. Pini, F. Leali, C. Secchi, and C. Fantuzzi, "Survey on human-robot interaction for robot programming in industrial applications," *IFAC-PapersOnLine*, vol. 51, no. 11, pp. 66–71, 2018.
[22] H. Canbolat, *Robots Operating in Hazardous Environments*. Norderstedt, Germany: Books on Demand, 2017.
[23] F. Dimeas, F. Fotiadis, D. Papageorgiou, A. Sidiropoulos, and Z. Doulgeri, "Towards progressive automation of repetitive tasks through physical human-robot interaction," in *Human Friendly Robotics*. Cham, Switzerland: Springer, 2019, pp. 151–163.
[24] K. Darvish, F. Wanderlingh, B. Bruno, E. Simetti, F. Mastrogiovanni, and G. Casalino, "Flexible human–robot cooperation models for assisted shop-floor tasks," *Mechatronics*, vol. 51, pp. 97–114, May 2018.
[25] K. Darvish, E. Simetti, F. Mastrogiovanni, and G. Casalino, "A hierarchical architecture for human-robot cooperation processes," 2020, *arXiv:2009.02807*. [Online]. Available: http://arxiv.org/abs/2009.02807
[26] J. Diprose, B. MacDonald, J. Hosking, and B. Plimmer, "Designing an API at an appropriate abstraction level for programming social robot applications," *J. Vis. Lang. Comput.*, vol. 39, pp. 22–40, Apr. 2017.
[27] K. Dill, "Structural architecture—Common tricks of the trade," in *Game AI Pro: Collected Wisdom of Game AI Professionals*. Boca Raton, FL, USA: CRC Press, 2013, p. 61.
[28] D. Garcia, L. Segars, and J. Paley, "Snap!(build your own blocks): Tutorial presentation," *J. Comput. Sci. Colleges*, vol. 27, no. 4, pp. 120–121, 2012.
[29] M. Resnick, J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. Silver, B. Silvermann, and Y. Kafai, "Scratch: Programming for all," *Commun. ACM*, vol. 52, no. 11, pp. 60–67, 2009.
[30] Google. *Blockly*. Accessed: Jun. 6, 2020. [Online]. Available: https://developers.google.com/blockly
[31] J. Huang, T. Lau, and M. Cakmak, "Design and evaluation of a rapid programming system for service robots," in *Proc. 11th ACM/IEEE Int. Conf. Hum.-Robot Interact. (HRI)*, Mar. 2016, pp. 295–302.
[32] I. Zubrycki, M. Kolesiński, and G. Granosik, "Graphical programming interface for enabling non-technical professionals to program robots and Internet-of-Things devices," in *Proc. Int. Work-Conf. Artif. Neural Netw.* Cham, Switzerland: Springer, 2017, pp. 620–631.

[33] D. Weintrop, A. Afzal, J. Salac, P. Francis, B. Li, D. C. Shepherd, and D. Franklin, "Evaluating CoBlox: A comparative study of robotics programming environments for adult novices," in *Proc. CHI Conf. Hum. Factors Comput. Syst.*, Apr. 2018, p. 366.
[34] J. P. Diprose, B. Plimmer, B. A. MacDonald, and J. G. Hosking, "A human-centric API for programming socially interactive robots," in *Proc. IEEE Symp. Vis. Lang. Human-Centric Comput. (VL/HCC)*, Jul. 2014, pp. 121–128.
[35] G. N. Yannakakis and J. Togelius, *Artificial Intelligence and Games*, vol. 2. New York, NY, USA: Springer, 2018.
[36] B. Ur, E. Mcmanus, M. P. Y. Ho, and M. L. Littman, "Practical trigger-action programming in the smart home," in *Proc. SIGCHI Conf. Hum. Factors Comput. Syst.*, Apr. 2014, pp. 803–812.
[37] L. De Russis and F. Corno, "HomeRules: A tangible end-user programming interface for smart homes," in *Proc. 33rd Annu. ACM Conf. Extended Abstr. Hum. Factors Comput. Syst.*, Apr. 2015, pp. 2109–2114.
[38] M. Colledanchise and P. Ögren, "Behavior trees in robotics and AL: An introduction," 2018, *arXiv:1709.00084*. [Online]. Available: https://arxiv.org/abs/1709.00084
[39] M. Dawe, S. Garolinski, L. Dicken, T. Humphreys, and D. Mark, "Behavior selection algorithms: An overview," in *Game AI Pro: Collected Wisdom of Game AI Professionals*. Boca Raton, FL, USA: CRC Press, 2014, pp. 47–60.
[40] Rethink Robotics. *Intera Software Platform for Industrial Automation*. Accessed: Jun. 1, 2020. [Online]. Available: http://www.rethinkrobotics.com/intera
[41] A. J. Champandard and P. Dunstan, "The behavior tree starter kit," in *Game AI Pro: Collected Wisdom of Game AI Professionals*. Boca Raton, FL, USA: CRC Press, 2012, pp. 72–92.
[42] E. Coronado, F. Mastrogiovanni, and G. Venture, "Design of a human-centered robot framework for end-user programming and applications," in *ROMANSY 22-Robot Design, Dynamics and Control*. Cham, Switzerland: Springer, 2019, pp. 450–457.
[43] I. Zubrycki and G. Granosik, "Designing an interactive device for sensory therapy," in *Proc. 11th ACM/IEEE Int. Conf. Hum.-Robot Interact. (HRI)*, Mar. 2016, pp. 545–546.
[44] M. Seraj, S. Autexier, and J. Janssen, "BEESM, a block-based educational programming tool for end users," in *Proc. 10th Nordic Conf. Hum.-Comput. Interact.*, Sep. 2018, pp. 886–891.
[45] E. Coronado, F. Mastrogiovanni, and G. Venture, "Development of intelligent behaviors for social robots via user-friendly and modular programming tools," in *Proc. IEEE Workshop Adv. Robot. Social Impacts (ARSO)*, Sep. 2018, pp. 62–68.
[46] E. Coronado, X. Indurkhya, and G. Venture, "Robots meet children, development of semi-autonomous control systems for children-robot interaction in the wild," in *Proc. IEEE 4th Int. Conf. Adv. Robot. Mechatronics (ICARM)*, Jul. 2019, pp. 360–365.
[47] A. S. Tanenbaum and M. Van Steen, *Distributed Systems: Principles and Paradigms*. Upper Saddle River, NJ, USA: Prentice-Hall, 2007.
[48] D. Kortenkamp, R. Simmons, and D. Brugali, "Robotic systems architectures and programming," in *Springer Handbook of Robotics*. Cham, Switzerland: Springer, 2016, pp. 283–306.
[49] PyPI. *The Python Package Index (PYPI): A Repository of Software for the Python Programming Language*. Accessed: Jun. 1, 2020. [Online]. Available: https://pypi.org
[50] D. Brugali and P. Scandurra, "Component-based robotic engineering (part I) [tutorial]," *IEEE Robot. Autom. Mag.*, vol. 16, no. 4, pp. 84–96, Dec. 2009.
[51] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: An open-source robot operating system," in *Proc. ICRA Workshop Open Source Softw.*, 2009, vol. 3, no. 3.2, p. 5.
[52] C. Crick, G. Jay, S. Osentoski, B. Pitzer, and O. C. Jenkins, "Rosbridge: ROS for non-ROS users," in *Robotics Research*. Cham, Switzerland: Springer, 2017, pp. 493–504.
[53] Vue.js. *The Progressive Javascript Framework*. Accessed: Jun. 1, 2020. [Online]. Available: https://vuejs.org
[54] C. Anderson, "The model-view-viewmodel (MVVM) design pattern," in *Pro Business Applications With Silverlight 5*. Berkeley, CA, USA: Apress, 2012, pp. 461–499.
[55] Microsoft TechNet. (2018). *Blackboard Design Pattern*. [Online]. Available: https://social.technet.microsoft.com/wiki/contents/articles/13215.blackboard-design-pattern.aspx
[56] N. Bevan, "International standards for HCI and usability," *Int. J. Hum.-Comput. Stud.*, vol. 55, no. 4, pp. 533–552, Oct. 2001.
[57] T. Issa and P. Isaias, "Usability and human computer interaction (HCI)," in *Sustainable Design*. London, U.K.: Springer, 2015, pp. 19–36.
[58] J. Johnson, *Designing With the Mind in Mind: Simple Guide to Understanding User Interface Design Guidelines*. Amsterdam, The Netherlands: Elsevier, 2013.
[59] D. A. Norman, "Design principles for human-computer interfaces," in *Proc. SIGCHI Conf. Hum. Factors Comput. Syst. (CHI)*, 1983, pp. 1–10.
[60] Google. (2020). *Material Design Guidelines*. Accessed: Jun. 1, 2020. [Online]. Available: https://material.io/design
[61] Apple. (2020). *Apple Design*. Accessed: Jun. 1, 2020. [Online]. Available: https://developer.apple.com/design/
[62] Vuetify. *VUE Material Design Component Framework*. Accessed: Jun. 1, 2020. [Online]. Available: https://vuetifyjs.com
[63] Enrique Coronado. *Open Rize Blockly Generator*. Accessed: Jun. 1, 2020. [Online]. Available: https://github.com/enriquecoronadozu/Open-RIZE-Blockly-Generator
[64] N. Leonardi, M. Manca, F. Paternò, and C. Santoro, "Trigger-action programming for personalising humanoid robot behaviour," in *Proc. CHI Conf. Hum. Factors Comput. Syst.*, May 2019, p. 445.
[65] S. Chitta, I. Sucan, and S. Cousins, "MoveIt! [ROS topics]," *IEEE Robot. Autom. Mag.*, vol. 19, no. 1, pp. 18–19, Mar. 2012.
[66] C. Abras, D. Maloney-Krichmar, and J. Preece, "User-centered design," *Bainbridge, W. Encyclopedia Hum.-Comput. Interact., Thousand Oaks, Sage Publications*, vol. 37, no. 4, pp. 445–456, 2004.
[67] C. Bartneck, T. Belpaeme, F. Eyssel, T. Kanda, M. Keijsers, and S. Šabanović, *Human-Robot Interaction: An Introduction*. Cambridge, U.K.: Cambridge Univ. Press, 2020.
[68] D. Deuff, I. Ocnarescu, L. E. Coronado, L. Rincon-Ardila, I. Milleville, and G. Venture, "Designerly way of thinking in a robotics research project," *J. Robot. Soc. Jpn.*, vol. 38, no. 8, pp. 692–702, 2020.
[69] L. Tian, P. Carreno-Medrano, A. Allen, S. Sumartojo, M. Mintrom, E. Coronado, G. Venture, E. Croft, and D. Kulić, "Redesigning human-robot interaction in response to robot failures: A participatory design methodology," in *Proc. CHI-Case Stud.*, 2020.
[70] L. Tian, P. Carreno-Medrano, S. Sumartojo, M. Mintrom, E. Coronado, G. Venture, and D. Kulić, "User expectations of robots in public spaces: A co-design methodology," in *Proc. ICSR*, 2020, pp. 259–270.
[71] GVlab. (2017). *Nao Bird and MC*. Accessed: Jun. 1, 2020. [Online]. Available: https://youtu.be/3H3aziEvMC4
[72] P. Carreno-Medrano, L. Tian, A. Allen, S. Sumartojo, M. Mintrom, E. Coronado, G. Venture, E. Croft, and D. Kulic, "Aligning Robot's behaviours and Users' perceptions through participatory prototyping," 2021, *arXiv:2101.03660*. [Online]. Available: http://arxiv.org/abs/2101.03660
[73] J. R. Lewis, "The system usability scale: Past, present, and future," *Int. J. Hum.-Comput. Interact.*, vol. 34, no. 7, pp. 577–590, Jul. 2018.
[74] T. S. Tullis and J. N. Stetson, "A comparison of questionnaires for assessing website usability," in *Proc. Usability Professionals Assoc. (UPA) Conf.*, 2004, pp. 1–14.
[75] Usability.gov. (2017). *Visual Design Basics*. Accessed: Jun. 1, 2017. [Online]. Available: https://www.usability.gov/what-and-why/visual-design
[76] A. Bangor, P. Kortum, and J. Miller, "Determining what individual SUS scores mean: Adding an adjective rating scale," *J. Usability Stud.*, vol. 4, no. 3, pp. 114–123, 2009.



ENRIQUE CORONADO received the B.S. degree in mechatronics engineering from the Autonomous University of San Luis Potosi, Mexico, in 2012, the M.S. degree in advanced robotics from the Ecole Centrale of Nantes, France, and also from the University of Genoa, Italy, in 2017, and the Ph.D. degree in mechanical engineering systems from the Tokyo University of Agriculture and Technology, Japan, in 2020. He is currently an Assistant Professor with the Department of Mechanical Systems Engineering, Tokyo University of Agriculture and Technology. His research interests include software architectures for robotics systems, human–robot interaction, end-user development, affecting computing, and artificial intelligence.

DOMINIQUE DEUFF received the Engineering degree in digital imaging from the University of Rennes 1, France, in 1997, the master's degree in ergonomics and applied her new skills to various projects from Orange, in 2008, and the Ph.D. degree in computer science from the University of Rennes 1, in 2003. She is currently pursuing the Ph.D. degree in ergonomics and design regarding social robotics at home with the University of Nantes. For 2 years, she was with the National Institute of Informatics, Japan as a Postdoctoral Researcher. She came back to France, in 2006, she joined Orange Labs as a Developer.

PAMELA CARRENO-MEDRANO received the combined B.Eng. degree in computer engineering from the École Nationale d'Ingénieurs de Brest, France, the M.Sc. degree in computer science from Universidad EAFIT, Colombia, in 2012, and the Ph.D. degree from the University of Bretagne-Sud, France, in 2016. From 2017 to 2019, she was a Postdoctoral Research Fellow with the University of Waterloo, Canada. She is currently a Postdoctoral Research Fellow with the Human–Robot Interaction Group, Faculty of Engineering, Monash University. Her research interests include understanding and analyzing human behavior for human–robot interaction purposes. In particular, she is interested in investigating how the underlying information humans convey through their movements and action choices can be used to inform and guide a robot's actions and social-interactive behavior. Her research interests include robot interactive learning, human–robot interaction, and affective computing.

LEIMIN TIAN received the B.Eng. degree from the Beijing University of Posts and Telecommunications, in 2012, and the M.Sc. degree and the Ph.D. degree in informatics from the Institute of Language, Cognition and Computation, School of Informatics, The University of Edinburgh, in 2013 and 2018, respectively. She is currently a Postdoctoral Research Fellow with the Human–Robot Interaction Group, School of Engineering, and also with the Human-Centered AI Group, Faculty of IT, Monash University. Her research interests include affective computing and human–robot interaction, in particular, combining human knowledge on emotions and the power of computational models to realize affective human–robot interaction. She has been serving as a Junior Committee Member for the Association for the Advancement of Affective Computing (AAAC), since 2018.

DANA KULIĆ (Member, IEEE) received the combined B.A.Sc. and M.Eng. degrees in electro-mechanical engineering, and the Ph.D. degree in mechanical engineering from the University of British Columbia, Canada, in 1998 and 2005, respectively. From 2006 to 2009, she was a JSPS Postdoctoral Fellow and a Project Assistant Professor with the Nakamura-Yamane Laboratory, The University of Tokyo, Japan. In 2009, she established the Adaptive System Laboratory, University of Waterloo, Canada, conducting research in human–robot interaction, human motion analysis for rehabilitation, and humanoid robotics. Since 2019, she has been a Professor and the Director of Monash robotics with Monash University, Australia. She conducts research in robotics and human–robot interaction, and develops autonomous systems that can operate in concert with humans, using natural and intuitive interaction strategies while learning from user feedback to improve and individualize operation over long-term use. Her research interests include robot learning, humanoid robots, human–robot interaction, and mechatronics. In 2020, she received the ARC Future Fellowship.

SHANTI SUMARTOJO is currently an Associate Professor of design research with Monash Art, Design and Architecture, and also a member of the Emerging Technologies Research Lab, Monash University. She collaborates with national and international colleagues in academia and the public sector, including industry-partnered projects, most recently in Australia with Exemplar Health, the City of Melbourne, and Lendlease. Her recent books include *Atmospheres and the Experiential World: Theory and Methods* (with Sarah Pink). Her research explores how people experience their spatial surroundings, including both material and immaterial aspects, with a particular focus on the built environment, design and technology, using ethnographic methodologies.

FULVIO MASTROGIOVANNI received the M.Eng. and Ph.D. degrees in robotics from the University of Genoa, in 2003 and 2008, respectively. He has been a Visiting Professor with the Asian Institute of Technology, Thailand; Shanghai Jiao Tong University, China; Karlsruhe Institute of Technology, Germany; Keio University, Japan; the Japan Advanced Institute of Science and Technology, Japan; and the Pontificia Universidad Catolica del Peru. He is currently an Associate Professor with the University of Genoa, Italy. He has published more than 170 articles in international journals and peer-reviewed international conferences, and one edited book. He served as a principal investigator or co-principal investigator in many funded projects at the national, European, and international levels. He is currently a member of the Admission and Review board of the start-up incubator Digital Tree. He is the Founder and Chief Research Officer of Teseo srl, a startup company working on human behavior monitoring, of IO Surgical Research srl, a startup company working on the adoption on wearable and holographic techniques in surgery applications, and of PHL srl, a startup company focusing on smart cities. He received three Best Paper Awards in international conferences. He served as a chair or co-chair for many international conferences and events, including IEEE RO-MAN 2013, 2015, 2016, 2017, 2018, and 2020, DARS 2014, URAI 2014, IEEE/RSJ IROS 2016, and ERF 2017. Until 2019, he has served as a Senior Editor for *Intelligent Service Robotics* (Springer). He is an Editor of *Robotics and Autonomous Systems* (Elsevier). From 2017 to 2019, he has been part of the Board of Directors of the Italian Association for Artificial Intelligence. His research interests include artificial intelligence techniques for robotics, cognitive systems, human–robot interaction, and collaboration.

GENTIANE VENTURE (Senior Member, IEEE) received the Engineering degree in robotics and automation from the Ecole Centrale of Nantes, France, in 2000, the M.Sc. degree in robotics from the University of Nantes, France, and the Ph.D. degree from the University of Nantes, in France, in 2003. In 2004, she joined the French Nuclear Agency, Paris, France, to work on the control of a tele-operated micro-manipulator. Later in 2004, she joined Prof. Yoshihiko Nakamura's Lab, The University of Tokyo, Japan, with the support of the JSPS. In 2006, still under Prof. Nakamura, she joined the IRT Project as a Project Assistant Professor. In March 2009, she becomes an Associate Professor and starts a new lab at the Tokyo University of Agriculture and Technology, Japan. Since July 2016, she has been a Distinguished Professor with the Tokyo University of Agriculture and Technology. Her current research interests include non-verbal communication, human behavior understanding from motion, human body modeling, dynamics identification, control of robot for human–robot interaction, and human affect recognition.