Tokyo University of Agriculture and Technology
Department of Mechanical Systems Engineering

Development of User-friendly, Distributed and Modular Robotics Frameworks for Enabling the Design and Integration of Social Intelligent Robots, Applications in Human-Robot Interaction

Ph.D. Thesis Report

Presented by

Luis Enrique Coronado Zuniga

Supervisor

Gentiane Venture

September 2020

Abstract

Robots, particularly social ones, are very promising machines that have the potential to enhance and complement many aspects of our social and everyday lives. However, the utility of these machines will remain restricted unless end-users (i.e., people who lack a robot engineering and information technology background) are able to design, execute, modify and update their own applications. A suitable way to enforce this possibility is through the End-User Development (EUD) paradigm. However, EUD for Robotics is a poorly addressed area outside Science, Technology, Engineering and Mathematics (STEM) aims. Moreover, the availability of robot software compatible with the resources (i.e., devices and software) of different types of users becomes relevant for supporting their research and work activities. However, many robotics frameworks enabling integration and application development are designed to exclusively or better work in desktop computers with some Linux-based desktop Operating Systems (OS), such as Ubuntu; therefore limiting their usability and accessibility. Furthermore, many academic and real-world projects can require high-performance, easy, and robust communication between software modules composed of a robotics system. These modules are often written in different programming languages or executed in different computers or smart-devices. This integration is often done to increase the robot's capabilities and functions using novel sensors and algorithms.

In an effort towards enabling end-users, novice programmers and robotics researchers to design and implement robot applications able to support their goals or needs, this thesis presents the NEP (Node Primitives) and RIZE (Robot Interfaces from Zero Experience) frameworks. On the one hand, NEP is a high–level and distributed robotics framework with an abstract model that encapsulates communication functionalities of most relevant robotics middlewares and general-purpose message libraries to enable the easy creation of advanced software architectures for robotics. On the other hand, RIZE is a EUD environment that enables the rapid prototyping of social robot applications. The first goal of NEP and RIZE is to provide reusable, flexible, robot-independent, accessible, cross-platform, and user-friendly development tools allowing people with a lack of advanced information technology skills to prototype and develop applications with robots. The second goal of NEP and RIZE is to enable multidisciplinary research on Human-Robot Interaction (HRI). For this, expert programmers can integrate new robot modules on the basis of end-users requirements. NEP is also designed to overcome limitations of existing distributed frameworks by enabling fast development and integration of robotics components which can be written in most relevant and modern programming languages and executed in several operating systems. Findings from comparative performance evaluations against state-of-art solutions prove the technological suitability of the NEP. Rather than only testing NEP and RIZE in laboratory tests, these tools are also tested in Human-Robot Interaction (HRI) applications performed "in the wild" (i.e., real-world applications executed in natural, crowded, dynamic and unstructured scenarios). Many of these applications were designed by interdisciplinary teams of robotics and social researchers being RIZE (which software architecture is based in NEP) a keystone software framework for reducing the gap between the different practices and domain expertise of researchers. Results from the different projects involved in this doctoral work prove the suitability of RIZE as a relevant and novel EUD tool for Robotics. Two group of end-user cases using RIZE are presented in this thesis: (i) long-term HRI in domestic environments and (ii) Children Robot Interaction (CRI) "in the wild". Due to their interdisciplinary nature, this thesis only reports those engineering aspects that are relevant for this doctoral work and not the findings of social researchers. Therefore, this thesis reports how RIZE was used by end-users for design and create their experimental scenarios with robots. Finally, NEP was used in several academic and end-user HRI projects, enabling undergraduate, master, and doctoral students on the GVlab to fast prototype advanced soft-

ware architectures for Robotics; therefore, supporting their research activities. Details of most relevant projects developed in the doctoral work are presented to prove the technological suitability of NEP for Robotics research. One of the most relevant applications is related to the creation of social intelligent emotional robots. These applications uses NEP for the integration of several sensory, perceptual, cognitive, and control modules written in different programming languages and executed on computers. Another relevant application was successfully presented in the International Robotics Exhibition (IREX) 2019. Results from applied questionnaires indicate that most visitors to this exhibition enjoyed the developed interactive scenario. Software frameworks presented in this doctoral work can be used as an initial step towards more user-friendly and reusable applications in Robotics. Applications shown in this thesis prove the technological suitability and potential of NEP and RIZE for empowering the development of both academic-oriented and real-world HRI projects.

Contents

List of Figures

List of Tables

List of abbreviations

AAI	Authoring Artificial Intelligence
ACE	Adaptive Communication Environment
AIST	Advanced Industrial Science and Technology
BDI	Belief-Desire-Intention
BT	Behavior Tree
CRI	Children-Robot Interaction
CSS	Cascading Style Sheets
CDF	Cognitive Dimension Framework
EUD	End-User Development
EUP	End-user programming
EUSE	End-user Software Engineering
FSM	Finite State Machines
HRI	Human Robot Interaction
HCI	Human-Computer Interaction
HTML	HyperText Markup Language
IoT	Internet of Things
IoRT	Internet of Robot Things
LfD	Learning from Demonstration
NARS	Negative Attitudes towards Robots Scale
NEP	Node Primitives
JSON	JavaScript Object Notation
PbD	Programming by Demonstration
ProCRob	Programming Cognitive Robot
RAT	Robot-Assisted Therapy
RIZE	Robot Interfaces from Zero Experience
ROS	Robot Operating System
RobAPL	Robot Agent Programming Language
SDK	Software Development Kit
SIR	Social Interactive Robots
SWIG	Simplified Wrapper and Interface Generator
SUS	System Usability Scale
TLX	Task Load Index
OS	Operating Systems
UX	User Experience
VPE	Visual Programming Environment
VPL	Visual Programming Language
WoZ	Wizard of Oz
YARP	Yet Another Robot Platform

Listings

Chapter 1

Introduction

1.1 Development of Social Intelligent Robots

Robots are advanced, useful and programmable machines nowadays successfully used in many manufacturing industries, universities, and research institutes. Traditional robotic structures (e.g., arm and parallel robots) are generally used to perform high-speed, efficient, tedious, and repetitive tasks in hazardous and industrial environments. In most of these scenarios, few or no interactive tasks with humans are required [5, 6]. These cases differ significantly from Socially Interactive Robots (SIR), whose main goals are to play useful social roles and engage different types of users through meaningful, natural, suitable, and safe interactions [7, 8]. A related concept is Human Robot Interaction (HRI), which is defined in [9] as "the field of study dedicated to understand, design, and evaluate robotic systems for their use by or with humans". The first HRI studies were performed using robots with a lack of sensing and reasoning capabilities in industrial and teleoperation tasks [10]. Then, the advances in sensing and computational capabilities allowed the development of novels methods of interaction with robots. Nowadays, interactive robots are not only aimed at expert users in laboratories or industrial scenarios. In fact, robots are also attempting to play several social roles such as helpers, companions, teachers, or friends in real-world scenarios. Despite being one of the most relevant emergent technologies according to the World Economic Forum 2019 [11], the successful adoption and acceptance of social and service robots into our society still requires the solution of many socioethical and technological challenges [12, 13]. Some of the most relevant socio-ethical issues are: privacy and security [14], legal issues [15], right levels of autonomy and anthropomorphism [16, 17], threatening of employments [18] and replacement of human interactions [15]. However, this thesis mostly focus in those technological challenges related to the development of applications and research activities with intelligent robotics systems, specifically on: (i) enabling the creation of advanced software architectures by the easy integration of software components for robotics; (ii) enabling interdisciplinary research activities where the owners of problems can develop their own applications and design desired intelligent robots; and (iii) bring robots to new, open, dynamic and natural scenarios. These three main challenges are described below.

1.1.1 Advanced Robot Programming

Distributed robotics frameworks [19] are the main software tools used in modern robotics to design and create advanced robotics systems. This is done by enabling the integration, execution, and management of several and independent software modules (also denoted as nodes), which represent the building block of any complex application in robotics. These modules generally address specific robotics tasks (e.g., data acquisition, perception, decision-making, and actuation). With the rapid development of sensors, devices, and algorithms in robotics,

nowadays new users outside its classical industrial and academic technological communities are motivated to develop novel robot applications able to solve their professional needs [20, 21]. To be adopted by a broader community and accessible to more people, robots must be able to communicate with user's devices (e.g., computers, tablets, smartphones, and smartwatches) for monitoring, management, programming and high-level control tasks. Enabling this connection using modern communication technologies is one of the main objectives of the Internet of Robot Things (IoRT) [22]. In this context, the availability of robot software compatible with the resources (i.e., devices and software) of different types of users becomes relevant for supporting their research and work activities. However, many academic-oriented component-based and distributed robot frameworks are designed to exclusively or better work in desktop computers with some specific Linux-based desktop Operating Systems (OS), such as Ubuntu [23]. This limits accessibility for many experts and novice users preferring or constrained to use other more popular OS. Moreover, many robotics projects can require high-performance, easy and robust communication between software modules written in different programming languages. This task is often done to increase the robot's capabilities and functions using novel sensors and algorithms. However, most distributed robotics frameworks tend to officially support a few programming languages, generally being C++ the main available option according to [23]. Furthermore, most robotics frameworks were originally designed for supporting academic projects; therefore, tending to have steep learning curves [24]. Consequently, novice programmers in robotics are required to acquire new skills through time-consuming training. One of the main focus of this thesis is to improve current state-of-art in distributed and component-based robotics frameworks by proposing a set of usable inter-process communication tools enabling the design and development of advanced software architectures for robotics. Main requirements of this novel framework are: (i) to work in many versions of Windows, Linux and Mac OSX, including those currently not fully compatible or still not well supported by most popular state-of-art distributed and component-based robotics frameworks; (ii) to enable easy integration of software modules written in most relevant and modern programming languages for robotics; (iii) be easy and simple enough for enabling a quick start and use by novice programmers; and (iv) provide acceptable communication performance for supporting advanced Human-Robot Interaction (HRI) applications.

1.1.2 End-User Development for Robotics

Similar to the early years of computing hardware, current social robotics applications and research are widely dominated by high-tech scribes (i.e., experts in programming or engineers [25]). Unlike the requirements posed by the introduction of robots in industrial scenarios, use cases in which robots must interact with people using social norms and conventions are better approached by user experience (UX) designers and researchers in social sciences [26]. As suggested by several works in HRI research, such as [27, 28, 29, 30, 31], "the role of domain knowledge specialists should not be neglected" [27]. In fact, the inclusion of this new type of users in the design processes of SIR is keystone to increase the quality, suitability and acceptance of intelligent systems and their applications [27, 30]. However, people belonging to this category are traditionally skillful in domains profoundly different from advanced robot and software development, and oftentimes lack the required level of expertise in advanced engineering topics, which are typically used to implement complex robot behaviors. Examples of approaches addressing this issue by enabling the inclusion of non-roboticists in the creation of interactive applications are user-centered [32] and participatory design [33]. Recent examples in Social Robotics applying these methods are presented in [34, 35]. As described in [36], user-centered and participatory design endorse the "design for use before use" paradigm, which requires a clear division of labour between the people assigned for the creation of applications

at design time and the people able to use and redesign the application at run time. According to [37], approaches requiring this division of labour have become problematic in many software development areas due to: i) a lack of expert software developers or (manpower) able to grasp and attend all possible users as well as their needs; ii) the dynamic change of requirements, which are often specific to individual domain applications; and iii) possible misunderstandings between expert software developers and their users due to the difference in backgrounds and practices.

End-User Development (EUD) has emerged as a suitable alternative to those approaches requiring a division of labour [37]. This is done by enabling novice users of computers and people without training on traditional programming languages, which are often denoted as end users, to redesign their own applications not only at design time but also at run time [38]. The goal of EUD is to evolve from *easy-to-use* to *easy-to-develop* interactive technologies [37]. This goal is not limited to software but also can include hardware artifacts, such as 3D printing [37]. A new and broad definition based in the meta-design manifesto [39] also considers EUD as a sociotechnical activity where users can develop all software and hardware systems that they use in their everyday life [40]; thereby, enabling the independence of the owners of the problems (i.e., end users) from the high-tech scribes [25]. The concept of EUD is related to End-User Programming (EUP) and End-User Software Engineering (EUSE). On the one hand, EUP is often considered as a sub-set of EUD [41], because it focuses only on the techniques used to enable end users to write their own programs, such as visual programming, domain-specific languages and natural language programming. In contrast, EUD not only focuses on the program creation phases but also on the methods and tools that are able to support the entire software development lifecycle [38]. This requires reaching independence from high-tech scribes during the use, redesign, configuration, and extension of the software and hardware artifacts [40]. On the other hand, EUSE takes a different approach compared to EUP and EUD. This is because EUSE mostly focuses on providing end users with solutions derived from traditional software engineering, such as debugging and version control, to promote the creation of high-quality software (i.e., reusable, reliable and efficient) [41, 42].

Recently, *Programming by Demonstration* (PbD) and *Learning from Demonstration* (LfD) have become in popular EUP approaches enabling industrial robots to perform manipulation tasks autonomously [43, 44], such as pick-and-place [45] and robotics assembly [43]. For this end users must use kinesthetic guidance, teleoperation or by using external controls to teach the possible movements robots can imitate to perform some tasks [46]. However, these approaches are rarely used for enabling complex social skills, such as communication and emotional intelligence. In contrast, *Visual Programming Environments* (VPEs) are EUD tools offering a good trade-off between usability (being easy to learn and easy to use), and the overall *complexity* characterizing the robot-based behaviors that can be developed with these tools. VPEs integrate a selected *Visual Programming Language* (VPL) to enable their users to create applications on the basis of such graphical elements as icons, blocks, arrows, forms, and figures, among others, rather than code only [47, 48]. However, literature reports few attempts for democratizing robot programming by enabling end-users to create robot applications using VPEs that run on top of component-based frameworks for robotics. This aspect allows for the integration of new robot platforms and the reuse of software modules available in the community. As described in chapter 5, most EUD and EUP tools for robotics using VPLs present many accessibility, usability and flexibility issues. Therefore, the second main goal of this thesis is to improve the current state-of-art in EUD for robotics by the creation of a novel EUD tool able to: (i) be accessible and usable enough for enabling the easy development of robot applications by end user; and (ii) be easy expanded or modified with novel perceptual and control algorithms.

1.1.3 Human-Robot Interaction "in the Wild"

Robotics is a field governed by short-term experiments performed in controlled and static scenarios, such as industries and laboratories [49]. This approach makes data collection more manageable and avoids many of the technical issues often presented when robots perform in open, uncertain and highly dynamic environments. However, the HRI community has recently expressed the necessity to move towards natural, open, everyday environments: an approach referred to as HRI "in the wild" [49, 50]. The importance of "in the wild" research is in the acquisition of more valuable quantitative and qualitative information, which can be used to improve the design of robots and their applications, therefore increasing their economic and social value [51]. Ideally, robots working "in the wild" must be able to adapt to their environment, interact with humans in a natural way and learn from these interactions. At the same time, robots need to expose suitable, ethical and explainable behaviors [52, 15, 53]. However, robotics systems presenting all these features can be very expensive (in time and resources) and difficult to develop even for expert programmers [54, 55]. To overcome some technical boundaries many works performed in laboratories and "in the wild" scenarios rely on Wizard of Oz (WoZ) or teleportation approaches [56, 57]. However, these approaches increase the cognitive workload of social researchers remotely operating robots [56]; therefore becoming unsuitable in application requiring long-term interactions [57]. An alternative to handling this problem is through the use of semi-autonomous systems (i.e., when robots controlled by both humans or some autonomous intelligence). Rather than only focus in laboratory settings, this work validates the suitability of proposed software tools for enabling research activities performed in different "in the wild " scenarios and with autonomous and semi-autonomous robots. The creation of HRI "in the wild" applications represent a bigger challenge than its counterpart implemented in controlled and structured scenarios, such as robotics laboratories. This is due to many technical issues that are difficult to anticipate and fix when working in public, natural and dynamic scenarios. However, this type of experiment can help researchers to better understand how people will react to robots in their daily-file environments as well as in the creation of more generalizable HRI theories [49]. For this, it is keystone the assembling of interdisciplinary research teams that span robotics, design, and the behavioral sciences, and in this way, to create more valuable experiments [49]. Therefore, the third focus of this thesis is to support interdisciplinary research teams from the technical and development standpoint by filling the gap between the classical and technological research activities in robotics and the behavioral and social sciences.

1.2 Motivations, Contributions, and Research Questions

The previous section briefly described current gaps in modern research activities towards the development of robot software aimed to be used and integrated by novice programmers, and end users as well as evaluated "in the wild" scenarios. Therefore, the main motivation or objective of this thesis is:

To increase the usability, accessibility, and flexibility of robot software with the creation of software tools that enable an interdisciplinary and platform-independent development of intelligent robotics systems, which can be able to successfully perform in laboratories and "in the wild" scenarios

From this general objective, three sub-objectives are defined bellow:

1. **(O1):** *Create a novel, open, usable, cross-platform and high-performance distributed and*

component-based robotics framework for enabling the easy design and development of advanced software architectures for robotics

2. **(O2):** *Create a novel Visual Programming Environment tool for supporting interdisciplinary research towards the End-User Development paradigm*

3. **(O3):** *Prove usability and technological suitability of proposed software tools in Human-Robot Interaction applications performed in laboratories and "in the wild" scenarios*

To address the objective *O1*, this thesis presents the Node Primitives (NEP) robotics framework, which is composed of a set of libraries and interfaces enabling inter-process communication capabilities for robotics development. To address the objective *O2*, this thesis presents the Robot Interface from Zero Experience (RIZE) a cross-platform EUD tool running on top of NEP, a usable VPL, and modern web technologies for enabling the easy creation of HRI applications by end users. Finally, in order to address objective *O3*, a set of experiments performed in open, natural crowded, dynamic and noisy scenarios are performed.

The following research question rose from proposed objectives as well as in the development of this thesis:

- **(RQ1)** Are existing distributed robotics frameworks user-friendly enough for supporting research and professional activities of end users and newcomers in Social Robotics?

- **(RQ2)** What VPE tools for the development of social and service robots have been proposed in the EUD and EUP literature to support end-users research goals or professional needs?

- **(RQ3)** What robot behavior modeling approaches have been used in these VPEs to enable the creation of intelligent robotics systems?

- **(RQ4)** What technologies, evaluation methods and software tools have been used by authors of these tools to develop these VPE?

- **(RQ5)** What are the open issues and challenges for VPEs in the domain of Social Robotics?

- **(RQ6)** Can NEP be used to create advanced software architectures written in different programming languages and/or different computers?

- **(RQ7)** Is the performance of NEP libraries suitable enough for supporting robotics projects requiring low latency communication between software components?

- **(RQ8)** Are the proposed software tools suitable for the creation and execution of HRI applications performed in "in the wild" as well as academic-oriented research projects?

1.3 Organization

This thesis has 2 main outcomes: NEP (a distributed robotics framework) and RIZE (a EUD tool). Therefore, the organization of the initial chapters of this thesis report is based on these two software tools. In these chapters, state-of-art and description of the proposed software frameworks are presented. The final chapters describe the most relevant research projects involved in this doctoral work. These projects are presented chronologically. A summary of the subsequent chapters composing this thesis is described below.

Chapter 2 aims to answer research question RQ1, by presenting advantages and drawbacks of most relevant distributed robotics frameworks.

Chapter 3 presents main technologies and approaches used in the development of the proposed software frameworks and user interfaces.

Chapter 4 presents the developed libraries of the NEP robotics framework as well as a performance study that proves the performance superiority of the proposed approach against the most relevant state-of-art solutions. Therefore, answering research question RQ7.

Chapter 5 addresses research questions RQ2, RQ3, RQ4, and RQ5 by presenting a systematic review of EUD and EUP tools for the creation of social and service robots.

Chapter 6 presents the design considerations, development details and main features of the final version of RIZE.

Chapter 7 presents the first big research project performed using NEP and RIZE. This project involves the use of a humanoid robot in a domestic environment for long-term HRI. Therefore, answering research question RQ8.

Chapter 8 presents how RIZE has been used to help real end users to design robotics applications that required to interact with children "in the wild" environments. This project also proves the suitability of developed software (research question RQ8).

Chapter 9 presents how the version of NEP for MATLAB/OCTAVE has been used to support academic-oriented application towards the creation of emotional intelligent robots. This chapter partially answer research questions RQ6 and RQ8.

Chapter 10 presents how NEP was used as the main software framework for enabling the creation of an emotionally intelligent robot. This robot interacted with visitors at an international exhibition. Proposed software architecture is composed of many modules written in several computers and executed in different computers. Therefore, completely answer research question RQ6 and RQ8.

Chapter 11 presents a summary of the discussion and limitations of the performed work as well as conclusions and possible research directions

Chapter 2

Distributed Robotic Frameworks

The development of advanced robotics systems requires expertise in several engineering areas (e.g., control theory, programming, mechanics, artificial intelligence, and computer vision). In addition, to improve the quality and user experience of applications using these robotics systems, is recommended to include end users and domain-specific experts in social sciences on design and development tasks [27, 30, 31]. Therefore, the creation of intelligent robotics systems from scratch is generally considered a very complex task that one single researcher can hardly address. This issue can be partially overcome by the use of a distributed robotic framework (also identified as "robotic middleware"), which are software tools aimed to help in the development of complex software architectures for robotics. This is done by enabling code reuse and integration with different types of software and hardware elements. Due to the importance of these types of software frameworks in modern robotics development (and for this thesis), this chapter aims to present user experience challenges of existing and most relevant distributed robotics frameworks. First, section 2.1 of this chapter describes relevant concepts about distributed software systems. Then, section 2.2 describes the methodology used to ask research question RQ1: *Are current distributed robotics frameworks user-friendly enough for support research and professional activities of end users and newcomers in Social Robotics?*. Section 2.3 briefly describes main features of most popular distributed robotics frameworks found in literature. Finally, section 2.4 briefly discusses and concludes about the analysis done in this chapter.

2.1 A brief introduction to distributed systems

As defined in [58] a distributed system is "a collection of autonomous computing elements that appears to its users as a single coherent system". This definition highlights two main features of distributed systems: (i) computing elements must perform independently of each other; and (ii) they need to collaborate to appear to be a single system. Distributed frameworks are software tools enabling the creation of distributed systems. The most basic task of a distributed framework is to provide a common infrastructure for enabling cooperation (i.e., communication) between the computing elements executed in a distributed system [58, 23]. These computing elements are often identified as *nodes*. The connection between nodes can be local (i.e., nodes in the same computer), remote (i.e., nodes in different computers connected via wireless or wired) or a combination of them. These nodes can be of many types and be used for several different tasks. They can also be written in many different programming languages. An essential software used to help in the creation of distributed applications is the middleware, which provides a set of services and tools for enabling communication between nodes. Figure 2.1 shows a simplified abstraction model of distributed systems based in [59, 58, 60, 61]. As shown in this figure, distributed robotics frameworks are placed on top of the

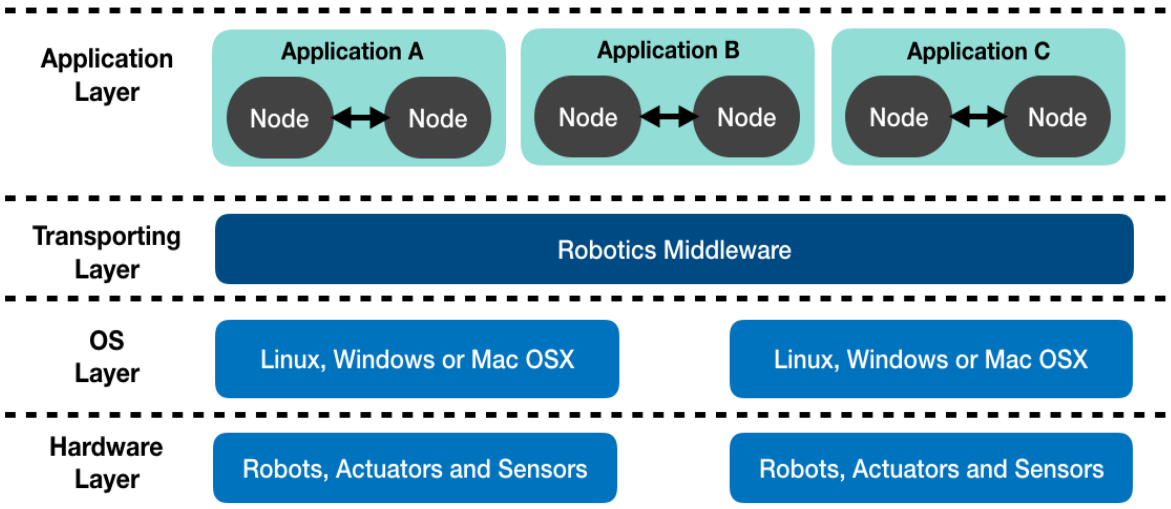


Figure 2.1: Abstraction model of distributed robotics systems

corresponding OS of the computing devices involved in the distributed system. Distributed robotics frameworks provide the required services and tools for enabling nodes involved in distributed applications to communicate between them, by hiding discrepancies of hardware and OS [58]. Figure 2.1 also shows typical examples of distributed applications. In application A and C nodes are executed in respective machine and communicated via the robotics middleware in a localhost network. In application B nodes are distributed in two different computers, which are remotely communicated using the robotics middleware. As described in [58] a distributed system ideally must: (a) enable interoperability between software implementations coming from different sources using a common standard (b) enable applications or interfaces developed in computer X to be executed, without modification or machine-dependent installation, on a different computer Y with the same of any other OS; (c) be easy to configure as well as easy to add new components or replace existing ones without affecting those components that stay in place, and (b) be transparent or invisible to end users and applications. However, in practice, these features can represent a big challenge [58].

Inter-process communication methods

A node can be seen as a group of protected resources (e.g., memory, open file descriptors, executable code, etc.). In a complex robotic system, several nodes must be able to communicate with others that run at the same time. The way in which nodes communicate is called inter-process communication. Classical forms of inter-process communication are signals, pipes, shared memory, and sockets. Signals are software interruptions that are generally used to notify the execution of some event to some process. However, due that signals are too slow and limited, they can be not sufficient to meet the need for high-bandwidth inter-process communication in many robotic applications. Pipes are file descriptors that allow two processes to communicate. They are more appropriate than signals to inter-proceed communication. However, they can be used only for communication between processes on the same machine. Moreover, it presents some inconveniences such as a limited buffer in non-blocking mode and a limited number of files that a process can open. When two or more processes have shared memory the information is immediately available to the other processes. However, access to the shared memory needs some synchronization primitives (semaphores, mutex, conditioned variables) in order to avoid corrupting the data. Shared memory is usually used in multithreading applications where

several threads or scheduling entities use the same address space in a process. Threads are considered lighter and faster to create that process. Also, the communication cost is generally lower between threads than between processes. Finally, sockets are a more general communication approach which is the standard for network programming in distributed systems. This approach is commonly used when communication between two processes on the same or different machines is needed. A socket is defined using an IP (Internet Protocol) address, a port that listens and a protocol. An example of endpoint definition is: $tcp://127.0.0.1:5000$ where *tcp* indicates the protocol, *127.0.0.1* indicates the IP address, and *5000* indicates the port of the connection.

Software connectors in robotics

A software connector in robotics can been seen as elements that encapsulate any details about communication protocols and synchronization. In [62] a robotic software communication synthesis denoted as "Protocol Stack View" (PSV) for distributed software was proposed. PSV describes design principles behind the contemporary distributed software frameworks. In base to PSV model and as suggested in [63], the of development of software connectors in robotics can be split in:

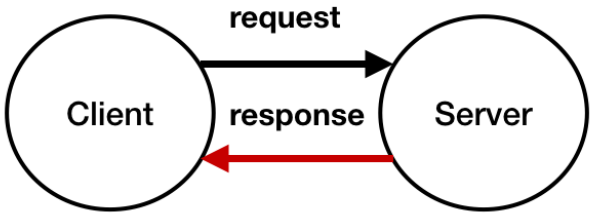
- **Transportation**: a collection of functionalities used to transfer data between process according to some set of rules. In this layer, the combination of the type of data and how these data are transferred is defined as a "network protocol". Examples of network protocols are the Transmission Control Protocol (TCP) and User Datagram Protocol (UDP).

- **Serialization**: defines how data structures and objects are converted and transmitted across a network through the transportation layer. Examples of serialization formats are XML (eXtensible Markup Language), JSON (JavaScript Object Notation), YAML (Yet Another Markup Language) and Protocol buffers.

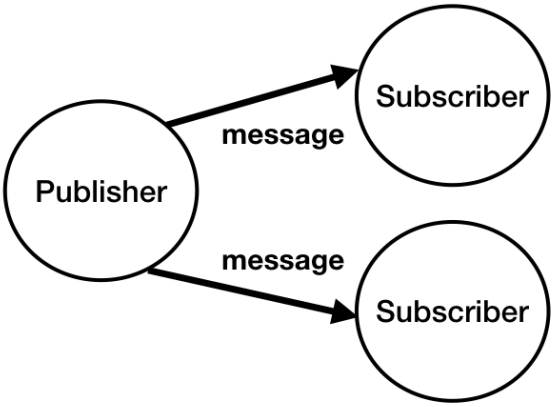
- **Service Discovery**: a collection of functionalities which allow distributed applications to register its services and to discover other services provided.

Messaging Patterns

Design patterns for information exchange are generally used to describe the flow of communication among processes in distributed systems. Two communication patterns widely used in Robotics are the *Request/Process/Reply* and the *Publish/Subscribe* models. One the one hand, the Request/Process/Reply model, also denoted as *Client/Server*, is one of the simplest communication patterns used by independent processes to share information and coordinate (Figure 3a). A process issues a request for some data and then goes to an idle state, waiting for the response to this request. If such a response is never received, a deadlock can occur. Although a non-blocking version of *Request/Process/Reply* exists, its use has non-obvious implications on the whole communication flow. On the other hand, in the *Publish/Subscribe* model the producers of messages, called *Publisher* process, send messages without the knowledge of what or if any receivers, called *Subscriber* processes, exist (Figure 3b). Unlike the *Request/Process/Reply* model, a *Publish/Subscribe* communication is typically asynchronous, which makes *Publish/Subscribe* more reliable.



(a) Request/Process/Reply



(b) Publish/Subscribe

Figure 2.2: More popular messaging patterns

2.2 Are distributed robotics frameworks user-friendly?

The main focus of the research question RQ1 is to discover if existing distributed robotics frameworks used in academia are user-friendly enough for enabling their easy use in research and professional activities of both end users and newcomers. However, user-friendly can be a subjective term that can depend on the expertise of users with computers and Unix-based systems. While many expert programmers and researchers preferring Linux distributions can consider the use of advanced approaches, such as using commands in a terminal, efficient and easy tasks, novice users of computers (most of them Windows users) can be very reluctant to use those expert-oriented approaches and feel more comfortable using graphical user interfaces (GUIs). Dimensions used to answer RQ1 are defined in table 2.1.

Due the main focus of this thesis is to support novice programmers and end-users, a robotic framework that is not user-friendly for the end user present the next issues: (i) requires to mainly use complex command lines for the installation of most core functionalities (in table defined 2.1 as RQ1-D1); (ii) requires to read complex documentation pages composed of many steps (RQ1-D2); (iii) requires to (re-)install the OS when updating to a new version of the distributed robotic framework (RQ1-D3); and (iv) requires the installation of complex third-party software for enabling the use of core functionalities (RQ1-D4). Therefore, in an ideal scenario, user-friendly installation and updating of a robotic framework must be performed using a unique, lightweight, familiar and consistent software wizard or setup assistant. Moreover, the core features of the robotics framework must be able to work without the installation or updating of complex or heavy third-party software. Furthermore, to be adopted by a broader community and accessible to more people, robotic frameworks must prioritize and provide full support popular OS for the general public (e.g,. Windows 7, Windows 10, Android, iOS and OSX). Dimension RQ2-D5 is used to show those issues of robotics frameworks when supporting popular OS. As described in [23], C++ is the dominant development language in most robotic

ID	Dimension	Description
RQ1-D1	Installation methods	Which aims to discover what are the approaches used to install basic packages or libraries of existing robotics frameworks in Windows
RQ1-D2	Simple-to-install	Which aims to discover if novice programmers and end-users can successfully install (in Windows) and quick start to using existing robotics frameworks without so much effort
RQ1-D3	Easy to update	Which aims to discover if users of these robotics frameworks can easily update the different versions of these frameworks
RQ1-D4	Third-party software	Which aims to discover if these robotics frameworks require the installation of some complex third-party software in Windows
RQ1-D5	Operating systems (OS)	Which aims to discover which OS are fully supported by these robotics frameworks
RQ1-D6	Programming languages	Which aims to discover which programming languages are officially and currently supported by these robotics frameworks
RQ1-D7	Pleasant and intuitive	Which aims to discover which robotics frameworks offers an intuitive, useful and aesthetically pleasant user interface for management and visualization of relevant data

Table 2.1: Dimensions used to answer RQ1

frameworks due to its benefits in speed and performance. However, many programmers prefer to use more usable, efficient, modern and high-level programming languages for fast prototyping of applications, such as Python, Java, Javascript, and C#, which are nowadays the most used in general programming according to [64]. Therefore, the native and official support of these more popular programming languages is keystone to broader robotics to more types of users and programmers as well as new information technology areas. Dimension (RQ1-D6) aims to discover which programming languages are officially supported by most relevant state-of-art robotics frameworks. Finally, a pleasant and intuitive GUI (RQ1-D7) is often described in Human-Computer Interaction (HCI) as required features to increase the perceived usability and general UX of users [65, 66].

2.3 Distributed frameworks for robotics

This section briefly describes the main features of more relevant state-of-art distributed robotics frameworks.

2.3.1 Robot Operating System (ROS 1.0)

The Robot Operating System (ROS), initially proposed in [67], is a mostly academic-oriented and popular robotics framework designed to support the development of applications in robotics research [68]. According to its official documentation [69], main features of ROS are: (i) enable message-passing between processes; (ii) enable programmers to build and run code across multiple computers; (iii) provides a package management system; (iv) provide mechanisms enabling hardware abstraction and low-level device control and (v) provides an implementation of some commonly-used robot functionalities for mobile and industrial robots (e.g. pose estimation algorithms, mapping, localization and navigation modules and trajectory planning). The primary goal of ROS is "to support code reuse in robotics research and development" [69]. Even when there is a clear dominance of ROS frameworks as the main communication tool for the integration of academic software in robotic laboratories, it is not totally free of disadvantages. As described in [70], ROS and most robot middleware still present many shortcomings regarding usability, portability, accessibility, compatibility and platform dependencies. The authors of [24] also describe the difficulty in learning as one of the main reasons why users do

not prefer or use ROS frameworks for their robotic projects. The main priority of most ROS releases has been to support Unix-based platforms, especially Ubuntu. Unfortunately, enabling complete and usable support of more popular and user-friendly OS (e.g Windows 7, 8 and 10) have been low-priority tasks for ROS developers and its community. This is observed in the documentation page of the latest release of ROS denoted as Melodic Morenia, which is mainly focused to be used in Ubuntu 18.04 [71]. As described in the official documentation in [72], targeted programming languages are C++, Python 2.7 and Lisp. As described in [23] the most famous graphical tools of ROS are *rviz* (for 3D visualization) as well as *rqt_graph* (for visualizing the ROS computation graph), *rqt_plot* (for visualizing numeric values in a 2D), *rqt_image_view* (for displaying images), which are based in *rqt* (a Qt-based framework for development of graphical interfaces for ROS). In order to use ROS, users require to acquire experience using command-line interfaces (rarely presented in *end users*) as well as deal with complex and many installations of third-party libraries (especially in Windows). In [73] creators of TiViPE [74], an state-of-art EUD tool for social robotics, have reported to critical issues of ROS for EUD: (i) most of the *end users* are Windows users and require easy-to-install tool; and (ii) *end users* will hardly understand (without training) many of the concepts required to use ROS.

To communicate with non-supported devices and programming languages, some robotics middleware/frameworks integrate bridge servers. As explained in Figure 2.3, this approach transforms middleware-dependent messages coming from middleware-dependent sockets to messages serialized in some format which non-supported devices or software modules can read and write. A popular serialization and data streaming format is the Javascript Object Notation (JSON). These messages are generally transmitted using the *Request/Process/Reply* communication pattern (also denoted *Client/Server*) via POSIX sockets or *Websockets*. The module that performs this transformation between protocols is often called a bridge server. Relevant state-of-art packages is the rosbridge suite [75]. As described in the official ROS documentation [76], rosbridge is composed of two parts: the protocol and the implementation. On the one hand, the rosbridge serialization protocol is a specification for sending JSON messages to ROS. On the other hand, the rosbridge implementation, also denoted rosbridge suite, is a collection of packages that implement the rosbridge protocol over *Websockets*. These packages include the rosbridge library, *rosapi* and rosbridge server. Any future reference to "rosbridge" in this thesis is made over the rosbridge implementation and not about the rosbridge protocol. However, the use of the *Client/Server* pattern tends to create less reliable software architectures due to communication issues (e.g., loss of connection and deadlocks) as well as lower performance for data streaming when comparing with the *Publish/Subscribe* pattern [77]. These issues can be trivial for short-term and structured experiments requiring the streaming of a low volume of data at relatively low rate. However, they become relevant when building complex HRI applications requiring: i) the streaming of high volume of data and sensory information at high rates, and ii) to be performed in long-term and unstructured Human–Robot Interaction (HRI) studies outside laboratories. Moreover, solutions such as rosbridge in ROS 1.0 force non-Linux users to acquire and configure an additional computer with Ubuntu to enable the execution of the bridge server from which all messages need to go through [75, 78].

2.3.2 ROS 2.0

ROS 2.0 is the new version of ROS designed to overcome relevant demands in the ROS community. Rather than only be aimed for purely research purposes, ROS 2.0 is also oriented to support production and industrial environments [79]. The main focus of ROS 2.0 is to enable real-time performance. Real-time computing is in many cases misunderstood with fast-performance processing or communication. Rather than prioritizing fast processing (e.g., increase frame rates), a real-time system must guarantee a response within specified time constraints, often

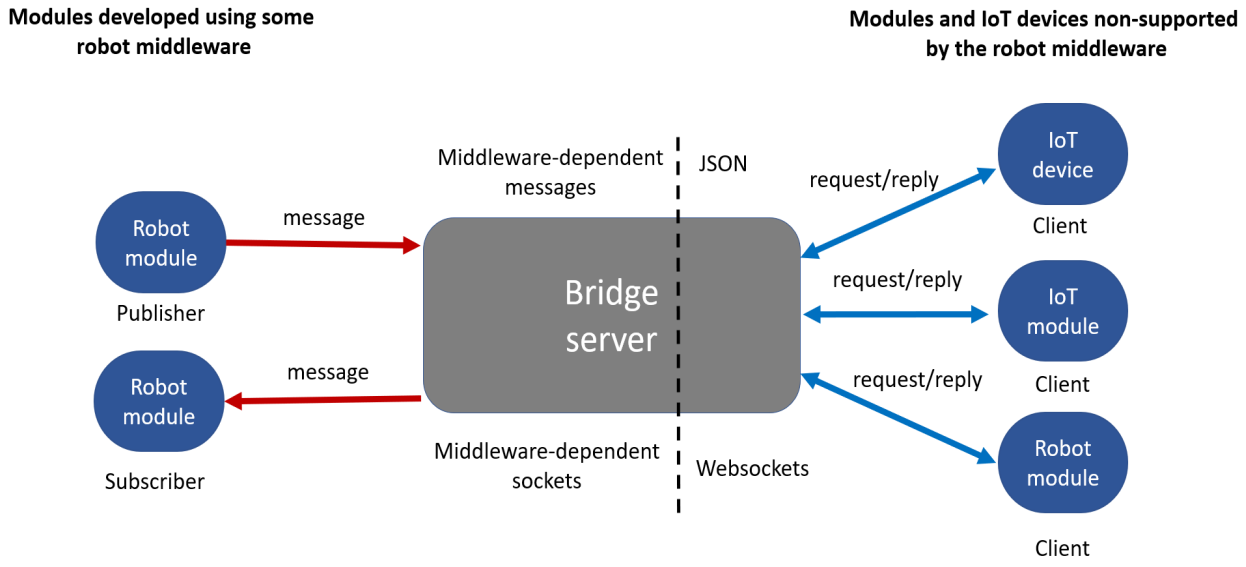


Figure 2.3: Example of robot and Internet of Things (IoT) components connected with non-robot-middleware-capable devices and modules via a bridge server

referred to as *deadlines* (e.g., ensure that the frame rate is stable). Targeted programming languages are C++, Python 3. Rather than be developed from scratch, ROS 2.0 is built on top of Data Distribution Services (DDS). Unfortunately, ROS 2.0 is still in its initial development *phases/releases*; therefore providing less usability and software tools when compared with ROS 1.0. In order to install ROS 2.0 in Windows, users must be experts on information technologies and computers. This is due to the fact that it is required to follow several complex and error-prone instructions in which users need to use an administrative shell, set environment variables in the control panel, and execute scripts using command lines. Therefore, the successful installation of ROS 2.0 can hardly be performed with success by novice end users of computers. Both version of ROS, 1.0 and 2.0, are dependent of the version of the OS (e.g. ROS 1.0 Kinetic is primarily targeted at the Ubuntu 16.04 and ROS 1.0 Melodic Morenia is primarily targeted at the Ubuntu 18.04) as well as third party libraries (e.g., ROS 2.0 requires visual studio 2015 and ROS 2.0 Eloquent Elusor requires visual studio 2019). This hinders the reusability of code as many ROS packages tend to be version-dependent. In order to use these version-dependent packages, ROS users often need to re-install a specific version of OS in their machines and deal with cumbersome and time-consuming installations of the required third-party libraries. As described in the official documentation of ROS 2.0 [80], "the primary platforms for ROS 2 are Canonical's Ubuntu releases", being Windows 10 supported in a beta version science it initial release in 2017. This can be seen as a signal of the poor attention and interest that currently have ROS developers and their community in the use of Windows-based systems as the main development environment. This can be due to the fact that typical developers and researchers using ROS (most of them expert programmers) prefer or are used to Linux systems. Another possible reason can be the lack of real-time capabilities of Windows 10 [81]. As described in [82] only Linux-based OS can take advantage of the real-time capabilities of ROS 2.0. The creation of distributed real-time systems is ideal for critical applications such as autonomous vehicles, spacecrafts, and industrial manufacturing and low tasks such as motion control and collision avoidance (which are out of the scope of this thesis). However, meeting hard real-time constraints is often considered technical overkill (i.e., the system is more high-performance and expensive to develop than what is required) for soft real-time systems (i.e, where occasional or

bounded deadline misses can be tolerated, as long as the process continuously receives required values) [83, 84]. Moreover, end users often work in high-level and non-critical layers, which can suitably be developed as required soft real-time systems.

2.3.3 Yet Another Robot Platform (YARP)

YARP is a cross-platform robotics framework developed by and for researchers in humanoid robotics [85]. YARP was developed in C++ and is mostly focused on providing high-performance communication features, such as reducing communication latency between nodes in a distributed software architecture for robotics. YARP is probably the most relevant ROS alternative and has been designed to be the main distribution platform for the iCub robot [86]. Unlike ROS 1.0, which middleware capabilities were developed from scratch, YARP was developed on top of the Adaptive Communication Environment (ACE) library [87, 88], from which inherits the portability to Windows, Linux, QNX 6, and Mac OSX [86]. Unlike ROS, YARP is not aimed at providing package management features nor a low-level build system macros and infrastructure such as ctakin [89]. Instead it was designed to be easy to interoperate with existing package management and building systems [85]. YARP can be installed in Windows using a setup assistant. However, it highly depends on the version of Visual Studio for code development and execution. YARP also provides a set of GUI developed in QT for data visualization, record data as well as running, stopping, killing and monitoring multiple programs on localhost or remote machines. However, these GUIs require to be launched and configured using command lines. In order to be used by other programming languages, YARP users must use the Simplified Wrapper and Interface Generator (SWIG). However, this task can be time-consuming and complex to use.

2.3.4 OpenRTM-aist

OpenRTM-aist is a development and middleware framework for component-based robotics. This robotic framework was developed in the Japan's National Institute of Advanced Industrial Science and Technology (AIST) based in the RT-Middleware [90]. The latest release of (1.2.1) OpenRTM-aist is able to run in Windows and Linux. This version is compatible with C++, Python 2.7, 3.6, and 3.7, and Java. This framework can be installed from a setup assistant and provides a large number of user-created software modules, denoted RT-Components. Two main tools of this robotic framework are *RT System Editor* and *RTC Builder*. On the one hand, *RT System Editor* is used for connecting RT-components and controlling the system architecture runtime [91]. However, this interface must be launched as plug-in Eclipse (an integrated development environment used in computer programming). On the other hand, *RTC Builder* is used for generating RT-components, which can be converted to an executable binary or a dynamically loadable shared library. OpenRTM-aist can be connected with OpenRTM.NET which is a .NET (C#, C++ and Visual Basic) implementation of the RT-Middleware. Unfortunately, a big portion of the documentation and available components are only in Japanese. This issue limits the accessibility of this framework for specific geographic areas.

2.4 Discussion

As observed in section 2.3 none of the most relevant distributed robotics frameworks satisfy all of the dimensions proposed in table 2.1. As described in sections 2.3.1 and 2.3.2 frameworks such as ROS 1.0 and ROS 2.0 are difficult to install and use in Windows machines (from the point of view of end users). They are also dependent on the OS; therefore, difficult to update. Moreover, robotics frameworks, such as YARP and ROS 2.0 require the installation of complex

and heavy third-party libraries. Furthermore, ROS 1.0 and ROS 2.0 are designed to exclusively or better work in desktop computers with some Linux-based desktop Operating Systems (OS), such as Ubuntu [23]. This limits accessibility for many experts and novice users preferring or constrained to use other more popular OS for the general user. Also, most robotics frameworks only support few programming languages by default, being C++ and Python the most used. Many of these frameworks also provide some GUIs for monitoring and management. However, these GUIs are developed using old development toolkits for creating interfaces and require the use of command lines for their execution and configuration. Limitations of the aforementioned robotics frameworks indicate that there is a need for a more user-friendly distributed robotics framework that end users can use transparently in many different types of computing devices with minimal effort. Solutions such as *rosbridge* have been developed to deal with some of these issues. However, they still need to be configured and launched by high-end scribes and often present poor communication performances. This last claim is proved in experimental evaluations performed in chapter 4.

Chapter 3

Software and Hardware

This chapter briefly described main technologies used in the development of proposed software frameworks and interfaces as well as those robotic platforms used to create HRI applications presented in this thesis.

3.1 ZeroMQ and nanomsg

ZeroMQ [92] and nanomsg [93] are an open-source, high-performance, asynchronous messaging libraries, from which developers can use for basic inter-process communication or develop their own middleware. ZeroMQ and nanomsg are socket library that provides several common communication patterns. These libraries aim to make the networking layer fast, scalable, and easy to use. They are also implemented in many languages, such as Python, C, C++, JavaScript, and C#. They also support all major mobile and desktop OS. Therefore, it can be used in the development of high-quality and platform-independent distributed systems. ZeroMQ offers a low-level programming library for the creation of several types of messaging patterns. This implies that the development and use of design patterns require more effort than other high-level middleware such as ROS. However, this also allows for more flexibility. Moreover, it lacks serialization and discovery services, which also imply a bigger effort in the initial development of a distributed system. Similar to ROS 2.0, which is built on top of Data Distribution Services (DDS) and YARP [85], which is built on top of the Adaptive Communication Environment (ACE) library [87, 88], The proposed distributed robotics frameworks adapt and simplifies the use to a low-level and general purpose communication libraries (ZeroMQ and nanomsg), and offers a set of supporting software tools for enabling the creation of advanced robot and IoT software architectures. This also includes the definition of a serialization format and the creation of a process enabling service discovery of nodes involved in a robotics system architecture. Examples of non-robotics middlewares built on top of ZeroMQ are presented in [94, 95, 96]. An approach enabling interfacing ROS and Unity using ZeroMQ is presented in [97]. However, the approach presented in this thesis is ROS-independent and focus to enable the support of most relevant and modern programming languages for general proposes programming.

3.2 Web-technologies

In order to enable the creation of platform-independent and usable applications, most of the proposed software interfaces are based on web technologies. Therefore, this section introduces the most relevant web-based technologies used in this thesis.

HTML, JavaScript and CSS

HyperText Markup Language (HTML), JavaScript and Cascading Style Sheets (CSS) are the core technologies in the World Wide Web. HTML is the standard language for creating web based applications. It is mainly used to specify the content of web pages. The web browsers can receive HTML documents from a webserver or from local storage and render them into web pages. Moreover, the HTML files can embed programs written in a scripting language such as JavaScript to specify the behavior of web pages. Finally, the inclusion of a CSS files is often used to define the look of the content of the web application. In order to use the next frameworks presented in this chapter is required to have an intermediate level of expertise in these three basic technologies.

3.2.1 Node.js

Node.js [98] is an event-driven and asynchronous Javascript runtime used to create server-side applications. This framework enables the execution of code Javascript code outside the web browser. This enables the easy creation of event-driven, asynchronous, modern and usable applications and user interfaces empowered by exiting web technologies. Moreover, Node.js present 4 key advantages for the objectives of this thesis: (i) An application developed in Node.js is intrinsically cross-platform; therefore the same code can run in Windows, OSX and Linux with very few modifications. (ii) Node.js is based in C++; therefore, enabling the creation of high-performance interfaces. (iii) Developed applications can be easily compiled by programmers as well as installed by end users without worrying about third-party libraries, which are installed automatically if any. (iv) It has a very big and growing community. Node.js also has a convenient package manager denoted as node package manager or *npm* which provides a very large variety of tools for developing advanced web-based applications. Listing 3.1 shows an example of a very simple package in Node.js.

```
{
    "name": "my_package",
    "description": "",
    "version": "1.0.0",
    "main": "index.js",
    "scripts": {
      "test": "echo \"Error: no test specified\" && exit 1"
    },
    "author": "",
    "license": "ISC",
}
```

Listing 3.1: Simple example of a Node.js package

Packages in Node.js makes ease for developers to manage and install software created in Node.js. Basic features often defined in a Node.js package are: the version, description, and name of the package, the main script of the package, a set of the command used to compile, execute, test, publish or distribute the package, the author name, and the license. Moreover, a list of packages or dependencies the project needs to be compiled and executed is often required. Code of the package configuration file for the creation of the nep.js library (one of the contribution of this thesis) is shown in appendix B.1. Package file enabling the execution, compilation and cross-platform distribution of RIZE (the EUD interface created in this thesis) is shown in appendix B.2.

3.2.2 Vue.js

Vue.js is a progressive framework for building user interfaces [1]. Vue.js enables users to render data to the HTML Document Object Model (DOM) in a declarative way. Data liked to the

(a) Before pressing the button (b) After pressing the button

Figure 3.1: Example of a simple GUI created with Vue.js, HTML and Javascript

DOM using Vue.js is reactive; therefore, if the data changes in the Javascript code, Vue.js will trigger an update of the interface to reflect that change. This Javascript framework uses the Model-View-ViewModel (MVVM) methodology [99], which is a software architectural and structural pattern that helps in the separation and development of the GUIs. For this the MVVM model separates the code required to build a GUI in three main groups: *Model*, *View* and *ViewModel*. The *Model*, which is the input or user data processed and allocated in the application memory. The *View*, which only includes the fronted or output of the application. Therefore, this group should not contain any logic or data manipulation tasks. The *ViewModel* bridges aforementioned groups by allowing the manipulation of data (in the *Model*) before it is output by the *View* [100]. An simple example of the *View* group in a user interface developed with Vue.js is presented in Listing 3.2. This group is developed in the HTML code. In this example a HTML container *div* has a id attribute with the value of *app*. This attribute is used for vue.js code in listing 3.3 to process outputs inside the container. In this case the HTML has two elements: text and a button. Text to be displayed is defined in the *message* variable and involved by double braces {{*message*}}. This creates a reactive variable that can changes dynamically by the code representing the *ViewModel* group. Moreover, an event is defined using the Vue.js statement *v-on:click* in the button. This relates the created event with the function *reverseMessage*. The *Model* group is expressed in the JavaScript Object Notation (JSON) inside the *data* element. In this case the variable *message* is defined a the only variable representing the *Model* group. The *ViewModel* group is expressed in the *methods* element, in which developers requires to define the events that will proecces the data displayed in the GUI. In this case the function *reverseMessage* will reverse the text displayed in the *message* variable, each time the button defined in the HTML code is pressed. The GUI created using this simple code is shown in figure 3.1.

```
<div id="app">
  <p>{{ message }}</p>
  <button v-on:click="reverseMessage">Reverse Message</button>
</div>
```

Listing 3.2: Example of code representing the View group in Vue.js

```
var app = new Vue({
  el: '#app',
  data: {
    message: 'Hello Vue.js!'
  },
  methods: {
    reverseMessage: function () {
      this.message = this.message.split('').reverse().join('')
    }
  }
})
```

Listing 3.3: Example of code representing the Model and ViewModel groups in Vue.js

Vue.js also allows building large-scale applications composed of small and reusable components, which organize a GUI into a tree of nested components as graphically represented in

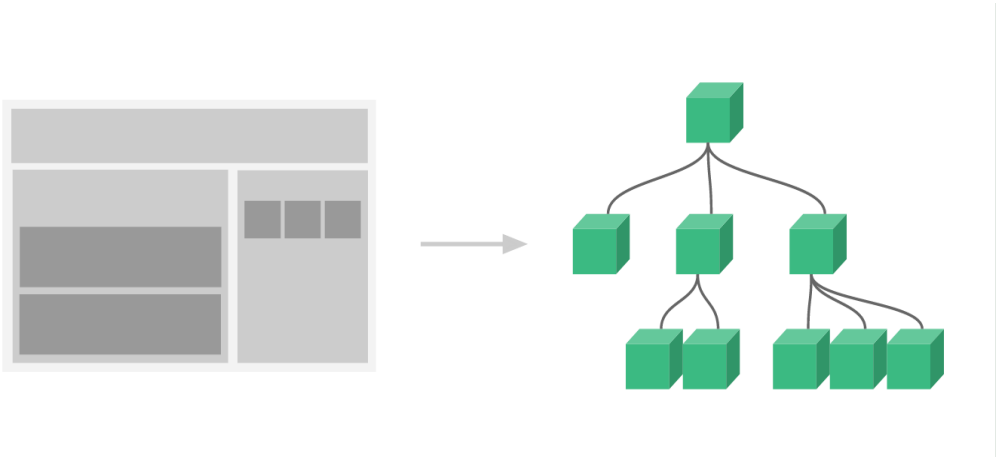


Figure 3.2: Graphical representation of a Vue.js application using components (from vue.js documentation[1])

figure 3.2. This approach was widely used in the development of the GUIs presented in this thesis. An example of a simple component in Vue.js is presenting in listing 3.4. Components can be defined by Javascript variables (in this case denoted as *example*). A Vue.js presents four basic sections: (i) *inputs*, which are the parameters that developers must entry to use the component;(ii) *data*, which represent reactive variable or *Model* of the MVVM methodology; (iii) *methods*, which enable data to react to user interactions; and (iv) *template*, which defines the HTML code required to define the outputs in the GUI.

```
var example = {
    // ------------- Input parameters ------------
    props: ["text_buton"],
    // --------------- Data model ------------------
    data: function () {
        return {
            dialog_robot_ready = false,
        }
    },
    // --------------- Functions -----------------
    methods: {

        reverseMessage: function () {
            this.message = this.message.split('').reverse().join('')
        }

    },
    // ---------------- HTML code ----------------
    template: `
    <p>{{message}}</p>
    <button v-on:click="reverseMessage">{{text_buton}}</button>`
}
```

Listing 3.4: Example of a simple component in vue.js

3.2.3 Electron

Electron [101] is an open-source library for Javascript and Node.js developed by GitHub for building cross-platform desktop applications with HTML, CSS, and JavaScript. Applications developed with electron can be installed from user-friendly desktop installers (i.e., .dmg, .exe, and .deb) in Windows 7, 8, 8,1 and 10, Mac OS X and recent versions of Linux. GUIs developed in this thesis use electron for enabling user-friendly installation and uninstallation of software.

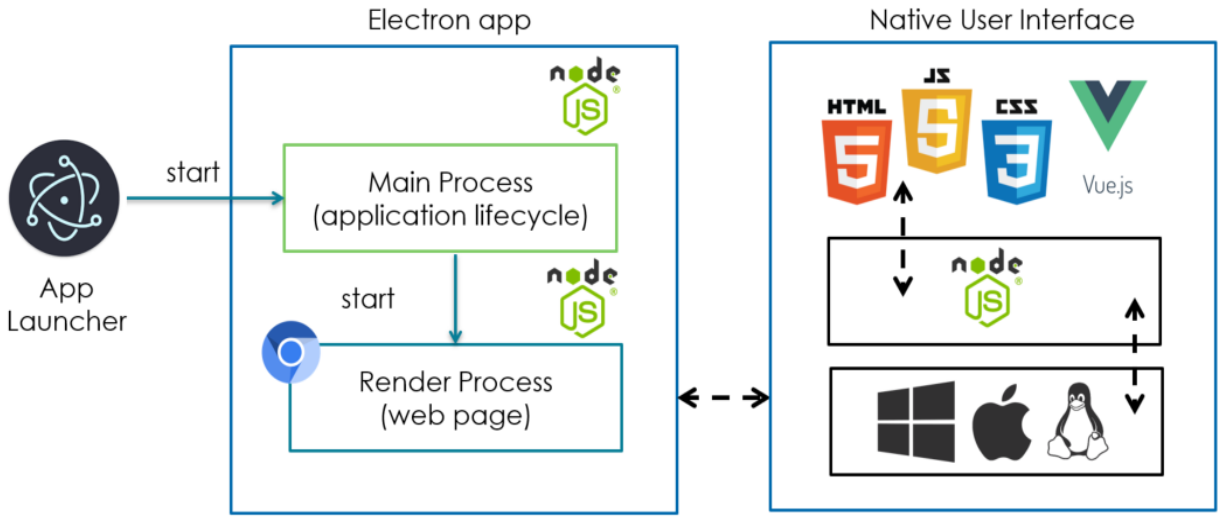


Figure 3.3: Software architecture of a Electron application

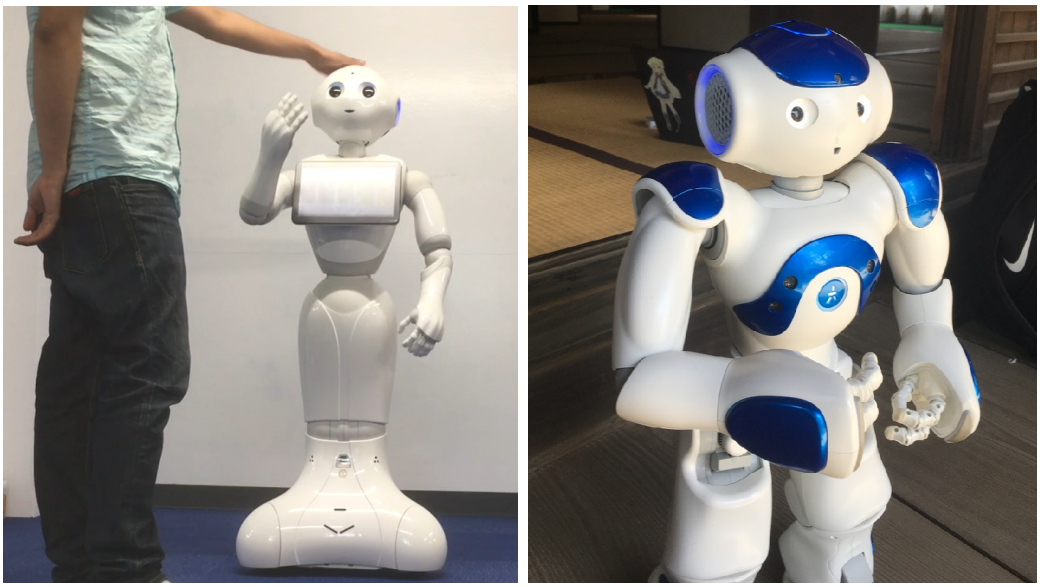
Figure 3.3 shows the software architecture of most user interfaces developed in this thesis using electron. Applications using electron are basically composed of a *main* process and one or more *render* processes. While the *main* process is in charge of creates and executes the *render* processes, the *render* processes create *Chromium* windows in which any web-based content can be executed. These electron processes can also access to system files and spawn other processes, such as Python scripts, using Node.js libraries. This approach enable the creation of usable and cross-platform Web-based applications that perform a Desktop applications.

Google Blockly

Blockly is Google's library for building visual programming editors based in an interlocking building blocks approach. This library is a refinement of Scratch, which is a popular software tool used to teach how to code to kids. Because is JavaScript and XML based, Google Blockly is aimed to be used in web-based applications. The main idea of this library is to allow the development of visual programming editors composed of several blocks. Each block can embed a variable, method or functionality which can be transformed into code of a real programming language such as Python and JavaScript. This software tool supports a wide range of modern web browsers such as Google Chrome, Firefox, and Safari, as well as iOS and Android devices. An example of a Google Blockly programming environment was shown in chapter 5 in figure 5.2. Basically a Google Blockly programming environment consists of a toolbox in which the user selects the block to use, a workspace where the user drag and drops the block to generate a program and a code generator tool in which it is possible to visualize the generated code in a specific programming language.

3.3 Robot platforms

Software frameworks developed in this thesis were designed to be platform-independent. However, this section briefly described those robot mostly used in the experimental validations presented in chapters 7, 8, 9 and 10.



(a) Pepper (b) Nao

Figure 3.4: Images of social humanoid robots used in this thesis

3.3.1 Nao and Pepper

NAO and Pepper are popular humanoid robot by Aldebaran/SoftBank Robotics. On the one hand, the current version of NAO has 25 degrees of freedom and have two HD cameras, two lateral speakers, four microphones, a sonar rangefinder, two infrared emitters and receivers, an inertial board, nine tactile sensors, and eight pressure sensors. It have also Ethernet and Wi-Fi connection. On the other hand, Pepper is a 1.20 metres and 28 kilograms humanoid robot nowadays also used in real world environments such as stores and airports. This robot has 20 degrees of freedom and have two HD cameras, one 3D sensor, two lateral speakers, four microphones on the head, 2 infrared sensors, tactile sensors, an internal unit and six laser line generators. Unlike NAO, Pepper also has a tablet allocated in its chest, which can be used to configure robot internet connection, execute application created by Aldebaran/SoftBank Robotics as well as be used to provide interactive and feedback capabilities to the users. Figure 3.4 shows images of the used Nao and Pepper robots in this thesis.

3.3.2 Robot arms controlled by MATLAB

An experimental robotics system controlled by MATLAB was used in several research projects focused on the development of affective social robots. The robotic system consists of two robot arm built with Dynamixel motors and several sensory devices (temperature, humidity, and brightness) to extract relevant information from the environment. Figure 3.5 shows the robot arms used.

3.3.3 Kawada Nextage

The Kawada Nextage, is a dual-arm industrial and collaborative robot designed to enable the advanced manipulation of objects. This robot has 15 degrees of freedom as well as a payload of 1.5 kg for each hand. It also has four uEye industrial cameras (2 in head and 1 in each hand), which can be connected to a computer using a USB 3.0 cable. This doctoral work uses the open version of this robot, which is compatible with the Robot Operating System (ROS) [67] in its Kinetic version. Low-level control and path planning tasks are performed in an Intel NUC (a

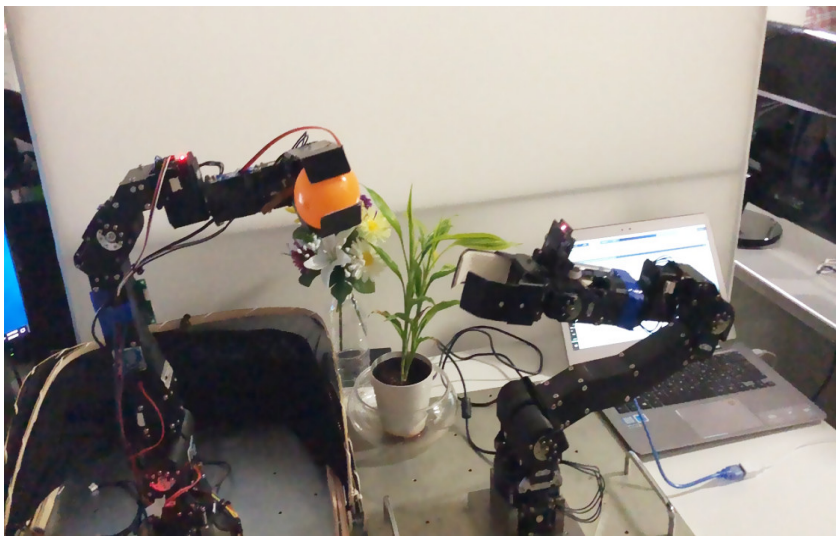


Figure 3.5: Robot arms controlled by MATLAB

small form factor computer) with Ubuntu 16 installed. The Kawada Nextage robot shown in figure 3.6 was used in this thesis for enabling HRI in an international exposition.

Figure 3.6: Nextage open collaborative robot

Chapter 4

The NEP Robotics Framework

Many robotics frameworks are available for enabling the creation of distributed software architectures. However, most of them present critical usability and UX issues hindering their use in research activities where end users and novice programmers must develop and use their own applications without the help of expert programmers and time-consuming training. This chapter presents the main features of NEP, a high-level robotic framework developed to address most of the issues presented in chapter 2. NEP provides a unified interface to a multiplicity of different robotics middlewares and general propose messaging-oriented libraries. NEP was developed for enabling easy and fast prototyping (through user-friendliness, modularity, and extensibility) by putting user experience front and center and by minimizing the number of user actions required for common use in the development of distributed systems for robotics. NEP is the core contribution this thesis.

4.1 What is NEP?

NEP is a high-level framework. Similarly to Keras [102], which is capable of running on top of TensorFlow [103] and Theano [104] for easy and fast prototyping of Deep Learning tasks, NEP is a high-level communication framework capable of running on top of low-level message libraries and robotic middlewares to provide simple and user-friendly development of accessible, re-usable and cross-platform robotics applications. This is done by enabling the communication between sensory, perceptual and cognitive robotics processes denoted as nodes (following the same terminology that ROS). These nodes, which are often written in different programming languages, can be executed on the same computer or in different computers and smart-devices connected at the *same cluster or IoT local network*. Unlike ROS, which is often used to handle low-level data generally defined in common structures or messages (e.g., force, torque, velocity, pose), NEP has been originally designed to deal with high-level data where there is not a unique or common way of representation (e.g. emotions, speech, gestures, behaviors, and objects). Therefore, messages in NEP are mostly defined using the JSON, which is a human-readable standard for the interchange of data. JSON is not only the preferred data interchange format for web services and applications, but also is highly used in many IoT applications [105, 106, 107]. However, NEP can also be used to send low-level and sensory data representations such as images.

NEP is a Robotics platform support. According to the definition classification done in [108] NEP is a IoRT robotics platform. Unlike ROS and YARP that provide a large variety of tools and algorithms for low-level development in robotics (e.g., simulators, inverse kinematic and planners), NEP focuses only on communication and the monitoring of data. This makes NEP easy-to-install using wizards (for *end-user* software distribution) or popular package manager (for developers), such as *pip*, *nuget* and *npm*. In the default configuration of NEP, links between

modules can be managed by a ROS-like Master node (more details in [67, 24]). This approach enables the creation of large, flexible and scalable applications. However, sockets connections (IP address and ports) in NEP can also be directly defined by the user, similar to YARP, for simpler and robust communication between modules. When reusing NEP code in ROS 1.0 and ROS 2.0 (i.e, selecting them as back-end options), links between modules are managed by the ROS Master and the installed Data Distribution Service (DDS) respectively.

NEP uses mainly ZeroMQ. The default back-end option of NEP is ZeroMQ [92] because it provides a lightweight, portable and high-performance library for inter-process communication that can be easily installed in almost all programming languages and OS. Furthermore, the creation of ROS-like peer-to-peer (P2P) communication using topics between components using ZeroMQ can be a difficult and time-consuming task requiring the manual configuration of sockets and the definition of the serialization approach (i.e., the process of encoding and transforming information sent via sockets from bytes to some data structure or object and vice-versa). This is due that ZeroMQ offer a very low-level API as well as a large variety of communication patterns and configurations; therefore, enabling their use for many engineering areas and types of software architectures. Similar to ROS 2.0 with DDS and YARP with the ACE library, NEP abstracts and simplifies the code required to use low-level libraries and robotics frameworks for enabling the easy creation of robotics software architectures.

NEP is cross-platform. NEP has been tested in Windows 10 and older Windows versions such as Windows 7 and Windows 8, which are still used by many industries and *end users* as proved in [109]. NEP has also been tested in other OS not fully supported by ROS frameworks such as older versions of OSX as well as older and newest versions of Ubuntu.

NEP focuses on human-centered projects. While NEP can be used for academic-oriented applications such as shown in this article, the main focus of NEP is to support *human-centered* design tasks, where non-roboticist are included in the design processes of robots. *Human-centered* approaches are intrinsically multidisciplinary tasks whose main focus is the development of accessible, desirable, intuitive, friendly and usable products able to satisfy human needs and expectations [110, 111]. In fact, most researchers in the robotics community mostly focus on *machine-centered* approaches being novelty and performance the main design objectives. As described in [110, 112], mature technologies nowadays adopted by the general public have historically switched their design approaches from *machine-centered* to *human-centered*.

NEP is not rosbridge. NEP does not provide a client implementation using the rosbridge protocol neither is a network protocol for exchanging JSON-encoded ROS topics over *Websockets*. Instead, the main communication protocol used in NEP is the ZeroMQ Message Transport Protocol (ZMTP) (described in [92]).

Current focus of NEP is not cloud robotics or web teleportation tasks. Cloud Robotics was initially defined in [113, 114] as "an approach to robotics that takes advantage of the Internet as a resource for massive parallel computation and real-time sharing of vast data resources". An example of a framework for cloud robotics is Rapyuta [115], which is used in the RoboEarth project [116]. Robot Web Tools [70] provides a set of open source modules using the rosbridge protocol as main technology to enable Client/Server messaging of ROS topics over wide area networks WAN. However, cloud robotics is out of the scope of the current objectives of NEP.

4.1.1 NEP for Python

Python is a popular programming language used in the development of Robotics and Artificial Intelligence projects. In many cases, the easy connection of different Python versions is required to enable the creation of intelligent robots. Due to this, NEP has been designed to work in both Python 2 and Python 3, using the same API. NEP for Python also provides a high-level abstraction communication layer which enables the easy switch between similar back-

end robotic middlewares and communication libraries. NEP is mostly oriented to support both *Client/Server* and *Publish/Subscribe* communication patterns. Current back-end options supporting the *Client/Server* communication pattern are POSIX sockets and ZeroMQ. Current back-end options supporting the *Publish/Subscribe* communication pattern are ROS 1.0, ROS 2.0, ZeroMQ and Nanomsg. The selection of the desired back-end is defined by the user..

To start using the basic NEP in Python, it is only required to write the command shown in Listing 4.1 in some console/terminal.

```
pip install -U nep
```

Listing 4.1: Install NEP in Python

As an example, the minimal Python code required to create a node publishing data and a node reading this data in NEP is shown in Listing 4.2 and 4.3 respectively.

```
import nep

node = nep.node('python_sender')          # Define new node
pub = node.new_pub('example','json')      # Define new publisher
msg = {'message':'hello'}                 # An example of message
pub.publish(msg)                          # Send message
```

Listing 4.2: Minimal code using NEP for creating a publisher in Python

```
import nep

node = nep.node('python_receiver')        # Define new node
sub = node.new_sub('example','json')      # Define subscriber

while True:
        s, msg = sub.listen()             # Read messages
        if s:                             # If there is a message
                print(msg["message"])     # Do something
```

Listing 4.3: Minimal code using NEP for creating a subscriber in Python

However, the script of Listing 4.3 will create a non-ROS re-usable subscriber. In ROS it is required to define special functions denoted as *callbacks* which are only executed when new data arrives at the socket connection. In order to enable the re-use of a NEP subscriber in ROS, the definition of a callback is also required. Minimal code enabling the definition of ROS re-usable callbacks functions in Python is shown in Listing 4.4.

```
import nep

# Callback defintion
def callback(msg):
        print(msg["message"])

# Define new node and the desired back-end (in this case ROS 2)
node = nep.node('python_receiver', 'ROS2')
# Set callbak to the subscriber
sub = node.new_callback('example','json', callback)
# Start event loop and block main thread
node.spin()
```

Listing 4.4: Minimal code using NEP for creating a ROS re-usable subscriber in Python

NEP was not designed to provide a mechanism that directly connects modules using different protocols and sockets types (e.g., a ZeroMQ subscriber can not read information from a publisher sending data using ROS). This feature is not even available in rosbridge, as messages sent and read from ROS modules to non-ROS compatible modules must be transformed to *Websockets* by the bridge server. Therefore, both the sender and receiver nodes must share the same middleware and protocol. However, users of Python can create their own glue modules

enabling the communication between different platforms and robot frameworks using the different back-end options of NEP. In Python NEP offers a high-level API that enables the re-use of code between different communication libraries and robot frameworks as well as simplifies their use and configuration for robot applications. Therefore, the same lines of code used to create a publisher or subscriber in NEP in Python 2 (which is supported by ROS 1.0 and ZeroMQ) can be reused in Python 3 (supported by ZeroMQ and ROS 2.0) just by changing the back-end option to use when creating a NEP node. In the example of Listing 4.4 the back-end option defined is set in line 8 as *'ROS2'* in the second parameter in the constructor of a new *nep.node* object. This parameter can be defined as *'ROS'*, to use this code in Python 2 and ROS 1.0 or *'ZMQ'* to use this code in both Python 3 or Python 2 using ZeroMQ sockets. When this parameter is not defined, such as Listing 4.2 and 4.3, ZeroMQ is used by default.

4.1.2 NEP for Javascript

The integration of JavaScript with robotic systems is relevant to enable the use of IoT technologies, such as cloud-based and web services as well as in the creation of usable Graphical User Interfaces (GUIs) providing better user experiences. NEP bindings for JavaScript and *Node.js*, denoted as *nep.js*, can be used to develop web-based applications able to be executed natively in desktop operating systems and visualized in web-browsers. NEP for JavaScript and *Node.js* can be easily installed writing the command shown in Listing 4.5 in some console/terminal. This library is keystone for the development of RIZE (the second main software outcome in this thesis).

```
1 npm i zeromq
2 npm i nep-js
```

Listing 4.5: Install NEP in Node.js

The minimal code required to create a publisher node and subscriber node in Javascript using Node.js are shown in Listings 4.6 and 4.7, respectively.

```
1 // Import main libraries
2 var zmq = require("zeromq");
3 var nep = require("nep-js");
4
5 var node = new nep.Node ("js_sender")           // Create a new ne
     node
6 var pub = node.new_pub("example", "json")    // Define publisher
7 ...
8 pub.publish({"message":"hello"})                // Send a message
```

Listing 4.6: Minimal code using NEP for creating publisher in JavaScript

```
1 // Import main libraries
2 var zmq = require("zeromq");
3 var nep = require("nep-js");
4
5 // Callback definition
6 var callback = function (msg) {
7     console.log(msg)
8 }
9
10 // Create a new ne node
11 var node = new nep.node ("js_receiver")
12 // Define subscriber
13 const sub = node.new_callback("example", "json", callback);
```

Listing 4.7: Minimal code using NEP for creating a subscriber in JavaScript

4.1.3 NEP for C#

C# is a developer-friendly programming language developed by Microsoft as an evolution of C++. Nowadays C# is one of the most popular programming languages, especially for game developing and data acquisition acquisition from sensors. Therefore, C# is often the main programming tool in many game engines, virtual and augmented reality systems and interactive sensors and devices (e.g., Cameras, Virtual Reality headsets, Kinect and Leap motion). The integration of many of these tools with robotic systems is often relevant for many researchers in robotics and biomechanics. A few examples can be seen in [117, 118, 119, 120, 121, 122]. In some cases, these tools are only officially supported or work better in Windows machines (in particular those products developed by Microsoft). NEP offers bindings for C# projects using Visual Studio. These bindings can be easily installed using *Nuget*, which is the official package manager for Microsoft development platforms. For this, users just require to write the commands of Listing 4.8 in the *Package Manager Console* of Visual Studio. For older version of Visual Studio and Windows OS (e.g. Windows 7) NEP for C# can be also added in the developer project adding the *AsynCIO*, *NetMQ*, *Newtonsoft* and *Nep* dynamic libraries in the project references.

```
Install-Package AsyncIO -Version 0.1.26
Install-Package NetMQ -Version 4.0.0.1
Install-Package Newtonsoft.Json
Install-Package Nep
```

Listing 4.8: Install NEP in C#

The minimal code required to create a publisher node and subscriber node in C# is shown in Listings 4.9 and 4.10, respectively.

```
class Msg // Message definition as a class
{
    public string message { get; set; }
}

static void Main(string[] args)
{
    // Create a new node
    Nep.Node node = new Nep.Node("csharp_sender");

    // Define publisher
    Nep.Publisher pub = node.new_pub("example", "json");

    // Define message
    Msg msg = new Msg();
    msg.message = "hello";

    // Send message
    pub.publish(msg);

}
```

Listing 4.9: Minimal code using NEP for creating publisher in C#

Because JSON objects in C# can not be implemented directly, NEP messages must be defined in a class, such as shown in lines 1-4 of Listing 9. In this example, the class *Mgs* is used to define the messages to send. In this class a JSON (*key*,*value*) pair is defined in line 3. The *key* element of the JSON pair will have the name of *message* and the *value* element will be a string. Then, the message to send is created as an object of the *Mgs* class and filled with the string *"hello"* in lines 15 and 16 respectively. Finally, the message is sent in line 19. Similar to the Publisher case, a Subscriber requires the definition of the message to receive as a C# class. Messages in NEP for C# are obtained as strings and must be converted to JSON values using the Json.NET library [123], as shown in line 19 of listing 4.10.

```
class Msg // Message definition as a class
{
    public string message { get; set; }
}

static void Main(string[] args)
{
    // Create a new node
    Nep.Node node = new Nep.Node("csharp_receiver");

    // Define subscriber
    Nep.Subscriber sub = node.new_sub("test");

    while (true)
    {
        // Get message as string
        string message = sub.listen();
        // Convert message to object
        Msg msg = JsonConvert.DeserializeObject<Msg>(message);
        // Print message
        Console.WriteLine(msg.message);
    }

}
```

Listing 4.10: Minimal code using NEP for creating a subscriber in C#

4.1.4 NEP for MATLAB and Octave

Nowadays many robotics and control researchers still consider the use of high-level numerical computing environments, such as MATLAB and Octave, as keystone software tools enabling prototyping, simulation, and validation of mechanisms, robot algorithms, and control systems. Moreover, many researchers use these numerical computing environments to perform data acquisition and perception task (e.g., computer vision and deep learning). In order to create more advanced robotics systems, programs written in MATLAB or Octave must be able to interface other external sensory, perceptual and cognitive modules in distributed systems. The most common way to enable the connection between two isolated processes is the use of conventional and synchronous TCP/IP BSD sockets [124, 125]. However, the use of conventional BSD sockets limits developers to the creation of *Client/Server* architectures. Compared with the *Publish/Subscribe* pattern, a *Client/Server* solution tends to have lower streaming data performance and creates less reliable software architectures [126, 77]. Some alternatives to conventional BSD sockets enabling the interfacing of MATLAB with external robotics modules have been proposed in the literature recently [127, 128]. Currently, the Robot Operating System (ROS) and Robotics System Toolboxes [127] were launched to enable the creation of ROS components in MATLAB. However, programs applying this approach can only communicate with other modules using the same version of ROS. A popular option used to connect different versions of ROS (i.e., ROS 1.0 with ROS 2.0) is the *rosbridge* suite [75]. This implementation of the *rosbridge* protocol has been used in many projects to connect programs using ROS with modules using programming languages or executed from operating systems not supported by ROS. On the other, BSD sockets alternatives supporting Octave are rarely reported in the literature. Unlike MATLAB, Octave is open source and free to use. The availability and accessibility of inter-process communication methods that enable the creation of distributed robotic systems in open-source platforms, such as Octave, is, in fact, a relevant contribution to the community. This can be especially valuable for those researchers, universities, students, and practitioners using Octave due to their personal preferences or license limitations (which makes the use of MATLAB a non-possible option).

NEP for MATLAB and Octave supports both *Client/Sever* and *Publish/Subscribe* communication patterns. As an example, the minimal code required to create a publisher node in NEP using MATLAB is shown in listing 4.11. Unlike versions for other programming languages and due some limitations of MATLAB when using binaries or libraries developed in JAVA, user requires to serialize the messages to send, which must be defined as MATLAB structures (e.g., line 10 of listing 4.11). This can be done in MATLAB using the *jsonencode* method as shown in line 11. Then, the converted MATLAB structure can be then published (line 12).

```
1  % -------------- Import NEP ---------------
2  import nep.Node;
3
4  % --------------- Create node  ------------
5  node = nep.Node("matlab_sender");
6
7  % ------------ Define publisher -----------
8  pub = node.new_pub("test", "json");
9
10 % Publish 5 times
11 for c = 1:5
12     msg = struct('message',c)    % Messages defined as structures
13     json_msg = jsonencode(msg)  % Convert matlab structures to JSON
14    pub.publish(json_msg)
15    pause(1)
16 end
17
18 % ----------------- Close publisher -----------------
19 pub.close()  % the subscriber must to be closed.
```

Listing 4.11: Minimal code using NEP for creating a publisher in Matlab

Minimal code required to create a subscriber able to read information from the publisher defined in listing 4.11 is shown in listing 4.12. Similar to publisher in MATLAB, subscribers requires an additional function for decoding the data obtained to a MATLAB structure.

```
1  % -------------- Import NEP ---------------
2  import nep.node;
3
4  % ----------------- New node ---------------
5  node = nep.node("matlab_receiver");
6
7  % ------------ Define subscriber -----------
8  sub = node.new_sub("test");
9
10 % --------------- Read data ---------------
11 while 1
12         msg = sub.listen(); % listen data
13         if strcmp(msg,"{}") % If message is null
14                 pause(.001)
15         else    %Covert JSON to matlab structure
16                 value = jsondecode(string(msg))
17         end
18 end
```

Listing 4.12: Minimal code using NEP for creating a subscriber in Matlab

A similar approach can be applied to generate a *Publish/Subscribe* architecture in Octave. These and more sample examples enabling *Client/Sever* and *Publish/Subscribe* architectures in both MATLAB and Octave as well as instructions for installing NEP tools and libraries are available in [129].

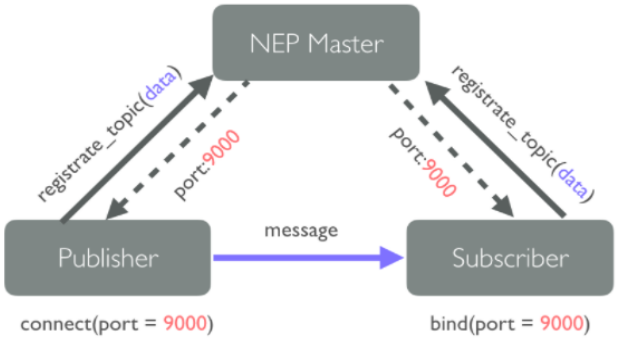


Figure 4.1: NEP Master architecture

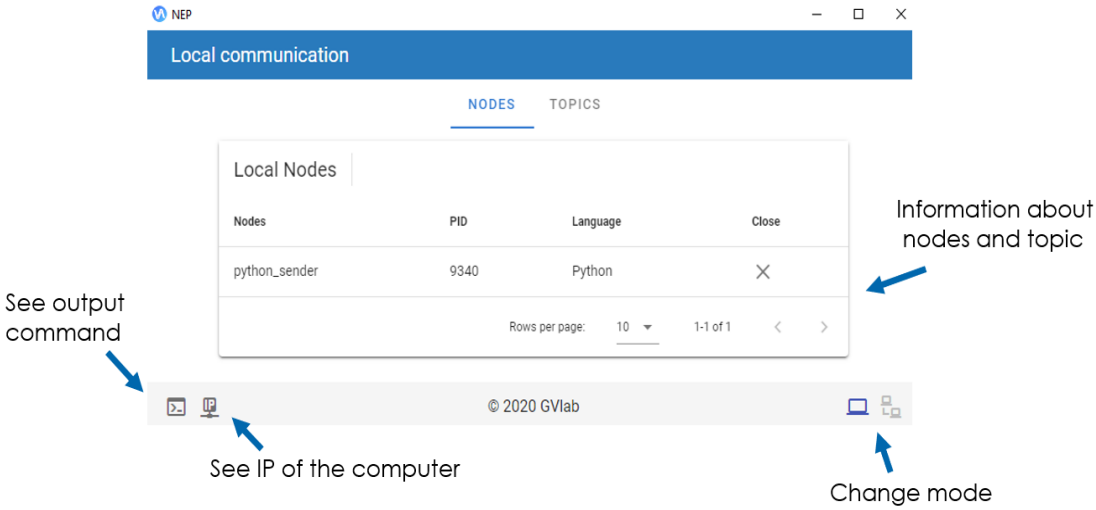


Figure 4.2: Main interface of the NEP Discovery Master Node

4.2 Discovery Service Master Node

The discovery problem can be defined as the process to find the other peers or nodes in the network. This problem is one of the main issues that developers must affront when large distributed architectures are designed. The simplest solution to this problem is using a set of fixed endpoints descriptions (IP address and ports) saved in strings or configuration files. However, this approach is unfeasible for large distributed projects due that increases the maintenance cost of the system. Another popular solution is to use a dynamic discovery node, which is a node or static point that manages the connections between all the nodes in a distributed system. The use of a master node reduces the number of endpoints to be configured to only one. The software architecture of the proposed master node is shown in figure 4.1. In this approach, the publisher and subscriber instances connect to the NEP master node using a *Client/Server* pattern. To connect nodes to the distributed system, the developer needs to specify the topics that will listen to publishing data in every node. Every topic needs to be registered in the master node, which manages the endpoint information of each topic. When a topic registration request is detected by the NEP master node, it assigns an endpoint direction to a specific topic. This endpoint information is saved and sent to the nodes that perform the request for that topic. Then, the transition of information is done when at least a pair of publisher-subscriber have requested the registration of the same topic.

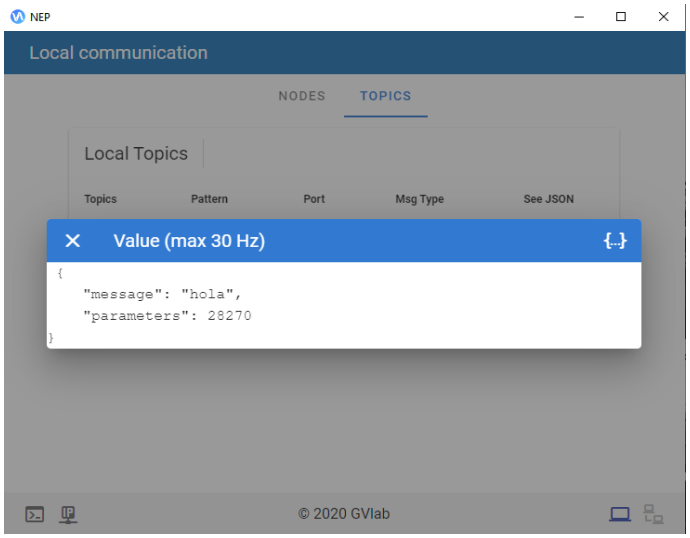


Figure 4.3: Example of data visualization using the NEP Discovery Master interface

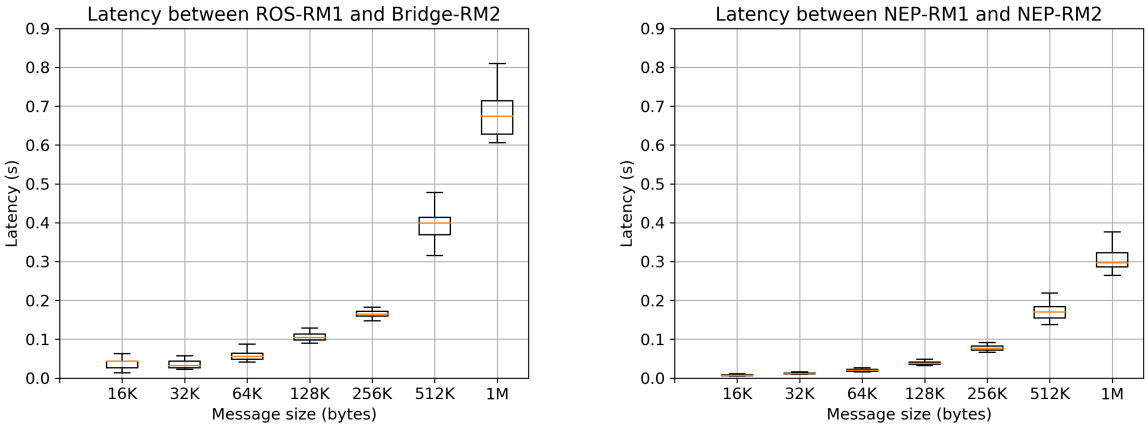
Rather than be launched using command line instructions, such as ROS and YARP, the NEP master node was designed as a web-based application. Therefore, end-users and developers can install this interface using a setup assistant in Windows and OSX. This interface can also be launched as any other user-friendly application in Windows (double-clicking an icon in the desktop or selecting the application in the taskbar of Windows). Users of MacOS X can use the launchpad for executing this interface. Users of Ubuntu requires double-clicking an executable. Figure 4.2 show current GUI developed for executing the master node. With this interface, developers can see the name of nodes and topics in a system architecture using NEP. Users can visualize information of nodes and topics that are working in localhost and in a remote approach (where a set of computers are connected using WiFi or Ethernet). They can also restart the master node which manage the remote connections, if the IP of the computer is changed. Other relevant information such as the current IP address of the computer and the output command in the scripts running the master nodes (in local and remote modes) can be also displayed. Moreover, uses can also use this interface for check messages send between nodes or computers as shown in figure 4.3. The current version of this interface only enable to display messages using the JSON format in a maximum of 30Hz. This limitation of 30Hz is done to avoid memory leak, which is presented when some information is very fast updated in a web interface executed in *electron* (a web-based technology used in this thesis for developing usable and cross-platform web-based interfaces). Details about the tools used to develop the proposed web interfaces in this thesis were briefly presented in chapter 3.

4.3 Performance Evaluation of NEP

This section present a comparative and performance study of NEP inter-process communication capabilities. The research question guiding this study was defined in chapter 1 as: *Is the performance of NEP libraries suitable enough for supporting robotics projects requiring low latency communication between software components?*.

4.3.1 Evaluation in Python and Node.js

In order to answer the proposed research question, a latency and comparative study against a state-of-art solution (the official rosbridge suite) was performed using Python and Node.js.



(a) Remote connection using ROS and rosbridge suite

(b) Remote connection using NEP (ZeroMQ)

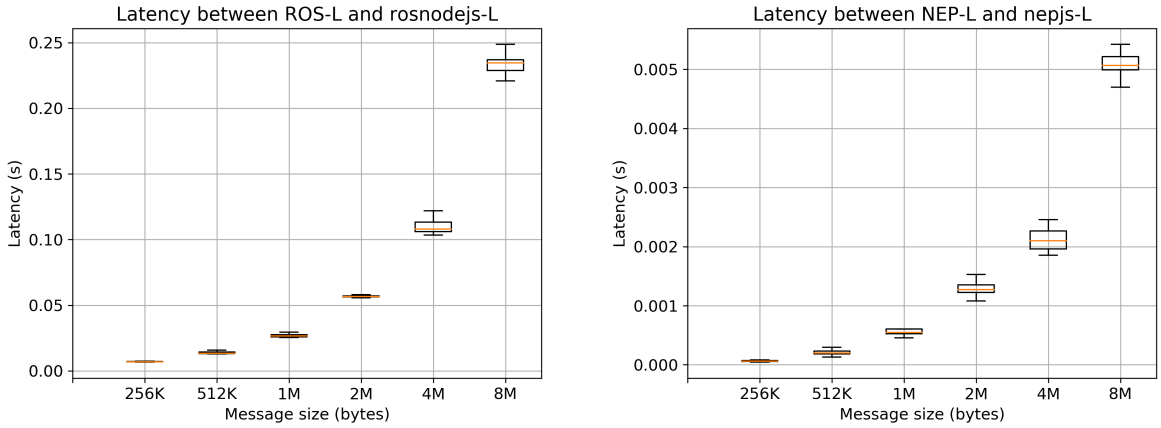
Figure 4.4: Latency results in scenario R-Remote; comparisons between NEP and ROS-rosbridge using nodes written in Python 2 and Python 3 executed in different machines, which are connected over the same Wifi network

This study focuses on two main scenarios: Remote communication in LAN and local-host communication of nodes. One the one hand, scenario (R-remote) is used to discover the suitability of NEP for messaging between 2 nodes launched in different computers connected to the same Wifi router. On the other hand, scenario (L-local) is used to discover the suitability of NEP for messaging between 2 nodes executed in the same computer but written in different programming languages. In this scenario, one of these nodes should be written in a programming language without the ROS official client library. For both scenarios, A-remote and B-local, I use string type messages of different sizes. The range of the transferred data size is 16 KB to 1 MB for R-remote scenario and 256K to 8M for the L-local scenario. A dedicated Wifi router where only the computers used in this study are connected must be used. Moreover, no additional processes must be executed in both computers as well as not an Internet connection must be done when executing the experiments. This is done in order to minimize latency due to external processes in these computers. For this study round-trip latency (i.e, time from the sender to the receiver in other programming language plus the time from the destination back to the sender) is measured.

Experimental Settings

Scenario R-remote is composed of two computers identified as *Machine1* and *Machine2*. The computer used for *Machine1* is a Laptop Dell Inspiron 14 with an Intel Core i7-7500U at 2.7GHz, 8 GB of RAM and Ubuntu 16 installed. The computer used for *Machine1* is a Laptop MSI GF63 8RD with an Intel Core i7-8750H at 2.2 GHz, 16 GB of RAM and Windows 10 installed. NEP 0.5.3.5, as well as ROS Kinetic with the official rosbridge suite, were installed in *Machine 1*. In *Machine 2* NEP 0.5.3.5 and the Python ROS Bridge library in version 0.7.1 were installed. Wi-Fi connection is performed using a low-cost and portable TP-link TL-WR802N router. Nodes in *Machine1*, which are identified as *NEP M1-1* (for a node using NEP) and *ROS M1-1* (for a node using ROS) are written in Python 2. Node in *Machine2*, which are identified as *NEP M2-2* (for a node using NEP) and *Bridge M2-2* (for a node using the Python ROS Bridge library) are written in Python 3. Communication is done in two cases: NEP-RM1 with NEP-RM2 and ROS RM1 with Bridge-RM2.

Scenario L-local was executed in the computer identified as *Machine 1* in the A-remote



(a) Local connection using ROS and rosbridge suite (b) Local connection using NEP (ZeroMQ)

Figure 4.5: Latency results on scenario L-local; comparisons between NEP and ROS-rosbridge using nodes written in Python 2 and Node.js executed in the same computer

scenario. Communication for this scenario is performed between nodes written in Python 2 and Node.js (Javascript). Nodes in this scenario are identified as *NEP-L* (for a node using NEP), *ROS-L* (for a node using ROS), *nepjs-L* (for a node using nep.js), *rosnodejs-L* (for a node using rosnodejs). Communication is done in two cases: NEP-L with nepjs-L and ROS-L with rosnodejs-L.

Results and Discussion

Results from comparative scenarios R-remote and L-remote are shown in Figure 4.4 and 4.5, respectively. These values are obtained after obtaining 200 samples for each message size. Figure 4.4 shows the case when two nodes executed in different computers are required to be connected. This figure clearly shows that the proposed approach using NEP over ZeroMQ as a back-end option outperforms latency results of ROS using the official rosbridge suite. Moreover, results in local-host of NEP highly outperform latency results reaching by ROS and the rosbridge suite. As mentioned in [130] communication using rosbridge to communicate ROS with other external modules "incurs in significant overhead due to the rosbridge transaction". With NEP this problem is highly reduced for both local and remote (in LAN) scenarios. These results ask the proposed research question, by proving the suitability of NEP framework for its use in academic-oriented projects.

4.3.2 Evaluation in MATLAB and OCTAVE

In order to prove the technological suitability of NEP for MATLAB/Octave, we compare its communication performance against relevant state-of-the-art solutions. For this task, the round-trip latency (time from the publisher in MATLAB/Octave to the destination in other programming language plus the time from destination back to MATLAB/Octave) is measured between: a) MATLAB and Python using ROS Toolbox, c) MATLAB and Python using NEP and b) OCTAVE and Python using NEP. It is relevant to highlight that the ROS toolbox is not compatible with Octave. These options of simulates cases when control programs written in MATLAB/Octave require to be connected with external software modules written in different programming languages and executed in a remote computer. In these evaluations, MATLAB 2019b and Octave 5.1.0 are executed in a Surface PRO 4 with an Intel Core i5-8350U with 8 GB of RAM and Windows 10. Remote PC running the external programs interfacing with

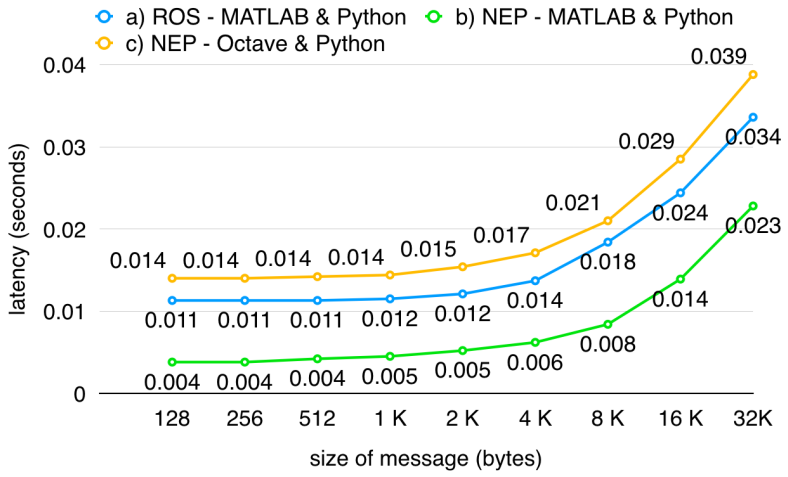


Figure 4.6: Latency comparison connecting MATLAB/Octave with Python using ROS Toolbox and NEP

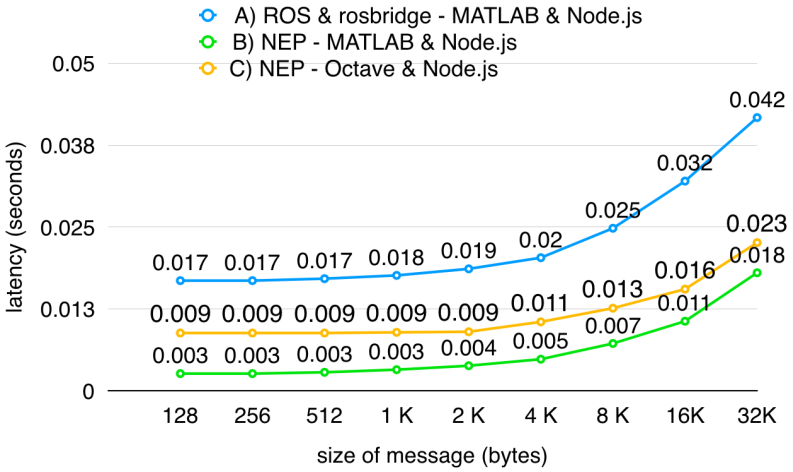


Figure 4.7: Latency comparison connecting MATLAB/Octave with Node.js using *rosbridge* and NEP

MATLAB are executed in a Laptop Dell Inspiron 14 with an Intel Core i7-7500U, 8 GB of RAM and Ubuntu 16. Wi-Fi connection is performed using a portable TP-link TL-WR802N modem for connecting these two computers.

To prove the suitability of using NEP for interfacing MATLAB/Octave with non-ROS enable software modules round-trip latency is registered between A) MATLAB and node.js using ROS Toolbox and the *rosbridge* suite, B) MATLAB and node.js using NEP and C) OCTAVE and node.js using NEP. Results for different message sizes are shown in figure 4.7.

Results and Discussion

Figure 4.6 shows the latency results for different message sizes executing the proposed scenarios. These values represent the mean of 200 samples for each message size. Results indicate that b) NEP in MATLAB highly outperforms latency results of a) ROS Toolbox in MATLAB and c) NEP in Octave. Result for communicating with Node.js also show lower latency using NEP and MATLAB. However, in this case the highest latency is reached when using *rosbridge* to communicate with Node.js. This can be due to the fact that messages must pass through a *rosbridge* server, which converts messages from ROS sockets to *Websockets*. Instead NEP enables direct communication between MATLAB/Octave and external modules written in many other programming languages. Moreover, an improvement in latency when interfacing with

Node.js is also obtained. This is an expected result as Node.js is built on top of C++, which is known to be of higher performance than Python.

Chapter 5

Visual Programming Environments for End-Users of Social Robots

In the past few years, a number of systems have been proposed that tackle EUD challenges for robotics systems at different levels, e.g., motion planning and execution frameworks adopting *Programming by Demonstration* (PbD) [131, 132], or the use of *Natural Language Processing* (NLP) to provide robots with instructions about how to carry out a certain task [133]. However, *Visual Programming Environments* (VPE) are still the EUP and EUD approach offering the best trade-off between usability (being easy to learn and easy to use), and the overall *complexity* characterizing the robot-based behaviors that can be developed with these tools. VPEs integrate a selected *Visual Programming Language* (VPL) to enable their users to create applications on the basis of such graphical elements as icons, blocks, arrows, forms, and figures, among others, rather than code only [47, 48]. The relevance of these development tools has recently increased not only in Robotics-related use cases but also in other fields of Computer Science, such as the *Internet of Things* (IoT) [48, 134] video game development [135, 136] mobile application development [137], and virtual/augmented reality [138]. Due to their aforementioned flexibility and relevance, this chapter focuses on those EUD tools using VPEs for the design of Social Robotics applications. For this a systematic literature review [139, 140] is performed. The organization of this chapter is as follows. Section 5.1 describes the needs and motivation for performing a systematic literature review of VPEs used for EUD of social and service robots. Section 5.2 presents some relevant concepts and guidelines used for the development and evaluation of programming tools for robotics. Section 5.3 describes the main concepts underlying VPE-based design, with an emphasis on Robotics-related requirements. Section 5.4 describes the methodology applied to perform a systematic analysis of the literature. Section 5.5 presents those VPEs tools found in literature. Section 5.6 presents the behavior modeling approaches used in VPEs for enabling the development intelligent social robots. Section 5.7 describes more relevant software tools and technologies executing at the back-end of these VPEs. Section 5.8 presents relevant open challenges, thereby proposing a road-map for further research directions. Conclusions are presented in the final section.

5.1 Motivations for performing a systematic review

As described by [38], most relevant literature reviews on EUD, such as [141, 142, 143], present a limited number of approaches and techniques. These relatively old literature review articles also tend to omit applications in Robotics. Moreover, the reviews presented in such articles have been performed and analyzed in the research domain of their authors [38], which differ from Robotics. Two more recent systematic reviews on EUD approaches are [144] and [38]. The authors of [144] conducted a 10 years (2007-2017) systematic search, in which 21 articles were finally selected. However, none of them are in the area of Robotics. In contrast, [38]

presents an overview of applications, tools and techniques in EUD, EUP and EUSE over 17 years (2000-2017). From the 165 papers finally selected in [38], only four are categorized in the Robotics area, of which only two are relevant for the focus of this article. Finally, a very recent narrative review of EUD-inspired software applied to domains such as smart homes, industrial and humanoid Robotics, task automation, and applications for human assistance are presented in [145]. However, the review proposed in [145] mainly focuses their analysis and proposed challenges to IoT-specific approaches and rule-based systems, such as the trigger-action paradigm [146]. In fact, most literature reviews on EUD tend to omit in their analysis the more recent tools, technologies, and approaches used in robotics aimed to enable the development of more advanced, reactive and robust robotic systems. Relevant methods often omitted are the behavior modeling approaches enabling end users to create intelligent social robots. These approaches, which are often denoted as *Authoring Artificial Intelligence* (AAI) or Artificial Intelligence (AI) architectures [147], enable the control of processes in which intelligent agents evaluate their environment to perform decision making. These AAI methods are nowadays widely used in areas where the development of complex and intelligent physical and virtual agents is needed, such as robotics and game development [30]. Three of the main research goals of AAI methods are: (i) overcome limitations in modularity, reliability, reusability, and robustness presented by the classical agent behavior modeling methods, such as rule-based systems and scripting [147]; (ii) enable their use for EUD and interaction design tasks [30]; and (iii) generate more intelligent, reactive, believable, suitable, and explainable agent behaviors [148]. Due to this, the role played by the AAI methods in EUD and EUP must not be omitted. Moreover, current systematic and narrative reviews in EUD also tend to omit in their analysis those relevant development practices and approaches nowadays used to create more advanced, reusable, scalable, interactive and reliable robotic systems. In particular are relevant those focus in the creation of distributed and modular software architectures, such as Component-based Software Engineering (CBSE) for robotics, which has become quasi-standard for modern robotic development [149, 77]. These frameworks have been described in chapter 2. Therefore, their analysis is fundamental for better understanding how to develop more advanced, usable and robust software back-ending EUD and EUP for robotics. Other recent reviews describing interesting intuitive programming tools, such as [150, 151], focus on educational contexts for children rather than EUD applications for adult users.

Limitations of the aforementioned studies indicate that there is a need for a more in-depth systematic identification of research articles enabling EUD for Social Robotics. This with the aim of providing a broader overview and understanding of the development approaches, tools, and practices enabling end users to create intelligent and interactive applications with social robots. To this end, this chapter presents a systematic search and analysis [139] of VPEs aimed at enabling the EUD paradigm for social HRI and everyday life applications.

5.2 Appropriate Abstraction Level for Programming Social Robots

Cognitive dimension is a framework used to analyze complex software tools such as programming languages and interactive user interfaces [152], [153]. It can be used to identify usability problems in early stages of the design of a user interface and to perform iterative design. A brief definition, based in [154] and [155], of the most relevant cognitive dimensions are shown in Table 5.1. As a usability principle, design of programming tools based in cognitive dimension must deal with a set of trade-off (i.e an attempt to improve some dimension always affects other dimensions). Therefore, cognitive dimension design must be goal oriented by selecting which dimensions are more important for the target audience.

Table 5.1: Cognitive dimension definitions

Dimension	Definition
Abstraction gradient	types and availability of abstraction mechanisms
Closeness of mapping	closeness of representation to domain
Hidden dependencies	important links between entities are not visible
Premature commitment	constraints on the order of doing things
Viscosity	resistance to change
Visibility	ability to view components easily
Diffuseness	verbosity of language
Error-proneness	notation invites mistakes
Hard mental operations	high demand on cognitive resources
Progressive evaluation	work-to-date can be checked at any time
Provisionality	degree of commitment to actions or marks
Role-expressiveness	the purpose of a component is readily inferred
Secondary notation	extra information in means other than formal syntax
Consistency	similar semantics are expressed in similar syntactic forms

Recently, [156] studied cognitive dimensions and usability trade-offs for the area of programming social robots. The analysis of [156] resulted in a proposal for a robot programming model that discomposes the social and intelligent abilities of robots in five abstraction levels: hardware primitives, algorithms primitives, social primitives, emergent primitives, and methods for controlling primitives. In this model, the lowest abstraction level is the hardware primitives that allow programmers to retrieve sensory information from hardware devices and control robot inputs (e.g., LED's and motors). The second abstraction level is the algorithm primitives that are used to build low-level interactive, perceptual and control capabilities in social robots, (e.g.,face tracking, sound source localization, and inverse kinematics). The third level of social primitives contains identified and reusable social interactive capabilities that are close to the domain expertise of the general end users. At the fourth level, emergent primitives are built from a combination of social primitives (e.g. gaze, speech, and gestures) to create very high-level social behaviors, such as emotions. Finally, the fifth level contains the control primitives that are in charge of performing decision making based on the current status of the interaction. The simplest way of doing this task is by using if-then-else rules. The description of methods used in EUD tools for social robotics for controlling primitives is the main focus of section 5.6. Findings of [156] suggest that the use of too many low-level abstractions (i.e. hardware and algorithm primitives) for the development of programming tools for social robots negatively affects their usability. Such low-level primitives tend to require hard mental operations and produce error-prone notations. At the same time, the use of too many emergent primitives affects the viscosity positively, but the expressive power of the programming tools and hidden dependencies negatively. In order to reach good usability and cognitive dimension trade-offs for the end user , these programming tools must use as many social primitives as possible. A deeper description of these primitives, and their influence on the usability and cognitive dimension trade-offs of a programming tool for social robots, is explained in depth in [156].

5.3 Visual Programming Environments in Robotics

With some simplification, three main Robotics-related scenarios can be identified where VPEs may play a decisive role, namely (i) industry settings, (ii) science, technology, engineering and mathematics (STEM) education, and (iii) end-user applications. Two major factors form a basis of this classification: the required programming abstraction level [156] and the target users.

The first distinguishing feature between these classes of use cases is based on the most

appropriate level of abstraction characterizing the programming *operators* and *primitives*, i.e., the available visual elements, which can provide the required usability, intuitiveness, ease of learning, and flexibility. While deciding to adopt low-level programming abstraction primitives can enhance the flexibility and the level of code reuse associated with VPEs, it also decreases their usability and intuitiveness [156]. Furthermore, the capability associated with easy-to-learn approaches can be negatively affected when developers must deal with a *mixture* of unbalanced abstraction levels [153, 156].

A second difference is the target end-users population in its own right. In use cases grounded in industry scenarios, the use of VPE-based approaches is geared towards the reduction of the costs associated with the development and maintenance of robot applications in the shop-floor or the manufacturing cell by human operators who are new or untrained in programming [157]. However, users of industry-oriented VPEs still require some expertise on low-level programming and Robotics notations. Examples of low-level notations presented in some VPEs for industrial settings are coordinate frames, tools, materials, joint velocities, end-effector orientations and positions, and hardware configurations. [158, 159, 160]. In general, industry-oriented VPEs are mainly focused on enabling robots to execute a set of well-defined sequences of repetitive and accurate tasks (e.g., assembly, pick-and-place, welding and material handling) [160], rather than enabling them to play complex and diverse social roles (e.g., teacher, friend or companion). Some advanced VPEs for industrial settings also enable a mixed approach with PbD methods [158, 160].

Use cases related to the adoption of VPEs in STEM and – in general – educational settings are typically aimed at children or new learners of general-purpose programming languages for developing *toy* programs rather than real-world applications. This type of VPEs is characterized by two main peculiarities. Firstly, they must be based on suitable approaches to enforce learning Computer Science or Robotics-related topics, such as the management of sensors or actuators, coding, functions, data structures, or algorithms. Secondly, these VPEs must be engaging and sufficiently easy-to-use to keep students interested and motivated during programming sessions. The abstraction level typically encoded in this type of interfaces depends also on the age of the target learners [161]. For elementary and middle school students, these software development environments must favour simplicity, intuitiveness and avoid the intrinsic complexity of general-purpose programming languages [162]. However, environments for students in high school and above often require the use of low-level general-purpose programming syntax (e.g., conditionals, loops and functions) to enable an easy transfer of knowledge to general-purpose programming languages or more complex approaches in advanced courses [163]. Nowadays, new STEM-targeted educational VPEs coupled with robot toys appear every year, notable examples being the interfaces for such robots as Cozmo [164] and Thymio [165].

Programming robots and learning how to use the available VPEs in both industrial and STEM-related educational scenarios are generally the main tasks or objectives of their target users. As mentioned above, these tasks require some low-level expertise in Robotics (in the case of industrial settings) or the user needs to acquire a complex body of knowledge by time-consuming training processes (in the case of STEM-related educational scenarios) [161]. This greatly differs from what is postulated by EUD approaches for domain-specific users [166, 143], in which *programming* is seen as an optional task to support work activities carried out by an end user rather than being a main learning or work objective. This may be the result of the fact that many domain specific end users do not have the time and motivation to learn how to use low-level Robotics software approaches [74, 167]. Therefore, VPEs for end users require more intuitive interfaces, mainly based on programming notations that are close to the domain knowledge of the general user. In many cases, these VPEs must also be flexible enough to enable the creation of complex, dynamic and engaging social interactive experiences with robots. Such interactions often require the use of multi-modal approaches, e.g., gesture

and speech recognition, expression of emotions, and engaging dialogues, among others [168]. Examples of domain specific end users are teachers developing robot tutors and helpers, artists programming a choreography or defining a script for robot-related artistic performances [21], sellers creating interactive experiences for customers [169], or therapists using robots to help in therapy sessions, just to name a few [74].

This chapter is targeted to discuss VPEs suitable to enable the adoption of EUD-based paradigms for the creation of social interaction applications by domain specific end users. According to the discussion in [170], most common VPL approaches can be categorized as: (i) form-filling, (ii) data-flow, and (iii) block programming. The description of these types of VPL is presented bellow.

Form-filling VPLs generally require the use of standard input forms, such as buttons and checkboxes, along with images to guide the user step-by-step in defining the trigger-action rules [170, 134]. While such VPLs are popular in different IoT environments, such as smart homes [170], they are very poorly explored in Robotics [171]. This can be due to the widely known limitations that these approaches present when are required for producing intelligent agent behaviors [30, 148]. Moreover, a lack of structure when defining disjoint *ad hoc* rules can make trigger-action systems unstable and error-prone when creating relatively complex programs, which requires the integration of additional tools for solving conflicts between rules [172, 148, 173]. Therefore, areas requiring the creation of more complex behaviors, such as video games and Robotics, prefer the use of more structured, robust, and expressive AAI approaches [147, 174].

Data-flow is a commonly adopted VPL approach in Robotics, not only for creating EUD-based environments for non-technically skilled people but also for expert use in complex and robust applications, as described in [175]. A *data-flow* programming environment is represented using directed graphs [176]. Nodes in *data-flow* interfaces are referred to by different terms by various authors, such as *blocks*, *functions*, *icons*, *states*, *procedures* and *boxes*. Nodes are connected by means of graphical *lines* (also cited as *wires* or *arcs*), which represent the flow of data between functions/blocks or transitions between states.

Block programming, based on the primitive-as-puzzle-piece metaphor (also known as block-based visual programming) [163, 177, 178], is a recently adopted approach that is gradually gaining attention in the development of EUD-based interfaces [178]. Unlike *data-flow* tools, visual elements in block-based VPLs are not connected using lines. Instead, block-based VPLs programs are built by assembling jigsaw puzzle pieces, which present visual cues that indicate to the user how visual elements may be used. This makes a block-based VPL an intuitive and engaging approach able to stimulate the user creativity [163, 170]. Following this definition, this review considers those VPEs using popular block-based VPLs such as Scratch [179], Snap! [180] or Google Blockly [181, 182].

5.4 Methodology

This work follow the guidelines proposed in [139, 140] for performing systematic reviews of Software Engineering. Systematic reviews are objective literature review studies used to identify relevant research papers, trends, gaps and challenges in some specific research area as well as to help in the position of research directions and activities [139]. Protocol for performing systematic review is based on recent and relevant systematic reviews with similar objectives and domain areas. Specifically, I followed [183], which focuses on the area of HRI, and [41] which focuses on the area of EUD. As described in [139, 140], and applied in [183], the process involved in a systematic review consists of five parts: (S1) definition of the *review protocol*, where the research questions are careful defined as well as the methods used to answer them; (S2)

ID	Dimension	Description and goal
RQ2-D1	Name	This dimension is used to identify each tools analyzed in this article
RQ2-D2	EUP approach	Which aims to discover the VPEs technique used to enable the creation of end-user programs. Values in this dimension can be *form-filling*, *data-flow* or *block-based*
RQ2-D3	Target users	Which aims to identify the type of end users these VPEs have been designed
RQ2-D4	Application domain	Which aims to discover the application domains in which these VPEs have been used to support end-users goals and needs
RQ2-D5	Target robot	Which aims to identify the type of robots supporting these VPEs

Table 5.2: Dimension used to obtain general information of VPEs

definition of the *search strategy*, which aims to identify the relevant research articles in the field; (S3) *documentation of search process*, where readers are able to evaluate how completely and rigorously the search process has been performed; (S4) specification of *inclusion* and *exclusion* criteria, which are used to select core articles in the field; and (S5) a report of relevant data or information from each research article or software tool.

5.4.1 Research questions

One of the main focus of this thesis is to complement recent literature review works in EUD, such as [41] and [145], by identifying and analyzing relevant tools and technologies integrating VPEs for enabling EUD and EUP in the Social Robotics area. The identification of these tools and technologies can be used to better understand the current scenario of EUD solutions for Social Robotics. Thus, the following research questions (also defined in chapter 1) were formulated.

(RQ2) What VPE tools for Social Robotics have been proposed in the literature to support end-users research goals or professional needs?

The dimensions used to respond to RQ2 are shown in Table 5.2. It is proposed RQ2-D1 (Name) to identify each tool resulting from the systematic search process. This enables a comparative analysis of these VPEs in the other research questions. Values of RQ2-D2 (EUP approach) are defined based on the VPEs classification presented in section 5.3. The formulation of dimensions RQ2-D3 and RQ2-D4 are based on those proposed in [41] to obtain general information of EUD, EUP, and EUSE tools. Therefore, RQ2-D3 and RQ2-D4 are focused on identifying the main target users and applications of these tools, respectively. Finally, RQ2-D5 is used to identify which social and service robots are supported by these VPEs. I propose these dimensions in order to get a general overview of the goals of relevant and recent VPEs for social robotics.

(RQ3) What robot behavior modeling AAI approaches, have been used in these VPEs to enable the creation of intelligent Social Robots?

RQ3 mostly focuses on: (i) how end users can effectively and intuitively compose programming primitives for the creation of their desired applications, and (ii) the methods enabling the control of these primitives. AAI approaches can be considered as those AI methods enabling the modeling and control of programming primitives used in VPEs for social robotics [156]. On the one hand, AAI for social robotics must be flexible and expressive enough; thereby, providing end users with an ability to create interesting and complex behaviors. On the other hand, they must be intuitive and simple enough for enabling easy creation and reuse of desired robot

ID	Dimension	Description
RQ3-D1	AAI approach	Which aims to identify the type of agent behavior modeling approach used for controlling programming primitives on reviewed VPEs
RQ3-D2	Programming primitives	Which aims to identify the type of programming primitives used in reviewed VPEs

Table 5.3: Dimension used to answer RQ3

behaviors. Therefore, the main goal of RQ3 is to identify the advantages and disadvantages of different AAI methods and how they have been used to enable the modeling intelligence on social robotics in VPEs. Dimension proposed to answer this research question are aimed to identify the used AAI approaches in VPEs for robotics (RQ3-D1) and identify the type of programming primitives generally used in these VPEs (RQ3-D2).

(RQ4) What technologies, evaluation methods and software tools have been used by authors of these tools to develop these VPE?

The focus of RQ4 is on the capabilities of proposed tools for enabling the independence of end users from high-tech scribes by supporting the end users in the entire life-cycle and not only in the creation phase. For this, EUD tools must be accessible, easy-to-use and install, support the end-user devices, and enable an easy extension of the software artifacts (e.g. addition of perceptual capabilities or re-use with other virtual or physical agents). Dimensions used to answer RQ4 are summarized in Table 5.4. Dimensions RQ4-D1 and RQ4-D2 are proposed to analyze the software approaches used to build VPEs for Social Robotics from which modular and reusability capabilities can be inferred. Dimensions RQ4-D3 to RQ4-D5 aim to discover which VPEs tools enable support of the entire life-cycle of application development. For this purpose, the user must be able to install, configure, and use these VPEs and create their own interactive scenarios with their robots in their own computing devices without the help of high-tech scribes. RQ4-D6 aims to identify the levels of liveness supported by these VPEs as well as simulator tools supporting them. Liveness is a concept used in literature for referring to the capabilities of programming systems to provide an immediate feedback cycle [184]. This feature can reduce the cognitive burden on programmers and enable users to adopt a more exploratory programming style [185]. According to [186, 187], it is possible to identify 4 levels of liveness: (i) level 1 (informative), where visual representations only understandable for expert developers are provided; (ii) level 2 (informative and significant), where visual representations of the programs have enough information to enable their execution; (iii) level 2 (informative, significant and responsive), where feedback can be provided on demand with a "run" button: and (iv) level 4 (informative, significant, responsive and live), where feedback is automatically provided as edits are done in the program. Unlike programs executed on a computer, robots can act and change the real-world environment. Therefore, feedback requiring the robot to perform motions requires special care. A safe option to implement level 4 of liveness is through simulations. RQ4-D7 aims to identify those methods used to evaluate the suitability of these VPEs. Finally, RQ4-D8 aims to identify which VPEs have been reported as: (i) only tested or evaluated by their developers and colleges (i.e. engineering students), (ii) used by real end users in design time (i.e., in laboratories); and (iii) used by real end users at run time (i.e, in the wild).

(RQ5) What are the open issues and challenges for VPEs in the domain of Social Robotics?

Research question RQ5 is mostly addressed in Section 5.8 based on the observed values of dimensions in RQ2, RQ3 and RQ4.

ID	Dimension	Description
RQ4-D1	Communication of modules	Which aims to discover if these VPEs have been developed using good practice for the integration of isolated robotics modules or nodes
RQ4-D2	Software technologies	Which aims to discover if these VPEs have been developed using modern technologies
RQ4-D3	Accessibility	Which aims to discover if these VPEs are online available; therefore end users can find them for their use and installation
RQ4-D4	Operating Systems (OS)	Which aims to discover the degree of support that VPEs have for the OS often used by end users
RQ4-D5	Easy-to-install and execute	Which aims to discover if VPEs can be installed and executed without the support of high-tech scribes
RQ4-D6	Liveness and Simulation	Which aims level of responsiveness of these VPes to the programmers edits as well available simulation capabilities
RQ4-D7	Evaluation methods	Which aims to identify which tools have been evaluated with real end users and which techniques were used in for these evaluations
RQ4-D8	Participation of end users	Which aims to identify the degree of participation that these tools enable their target end users (i.e., design time, use time or both)

Table 5.4: Dimensions used to answer RQ4

5.4.2 Search Process

The search was executed in well-established databases in the field of intelligent robotics systems: IEEE Xplore, Science Direct, ACM digital library, Springer Link and Web of Science. Examples of other systematic reviews focusing on robotics applications and methods using these sources are [183, 188, 189].

The time period of publications covered is between 2008 and 2018. The year 2008 was chosen as the starting year as no earlier tools are described in [37] and [41]. Moreover, 2008 is just before two major events occurred in Robotics, which are relevant for the focus of this article. One is the initial release of ROS [67] in version 1.0 (2009), which become a milestone in academic robot development. Approaches using ROS-like frameworks are nowadays quasi-standard for many researchers in the area of intelligent robot systems. The other is the release of the first public version of Nao social robot (2008) and its official EUD tool [190]. Nao is probably the most successfully used Social Robot up to date, which is evidenced by the fact that most of the VPEs reviewed in this article integrate this robot.

To obtain key terms for the search string, I applied three different strategies: (i) an analysis of the main goal of this chapter and the research questions, (ii) an analysis of core articles from previous state-of-the-art studies, and (iii) pilot testing. For step (i), the goal of this chapter is to identify relevant and recent *End-User Development* or *End-User Programming* supporting *Visual Programming Languages* for *Social Robotics*. These keywords are also contained in the research question (RQ2). Therefore, it is extracted *End-User*, *Programming*, *Visual*, *Development* and *Robot*. In step (ii), it is used the SEOBook keyword density analyzer [191] to identify the most recurrent words in two well-known core papers of EUD for Social Robotics, specifically [74] and [190]. Based on this analysis, keywords, such as *Robot*, *User*, *Development* and *Programming* are found to be relevant. In step (iii), I execute and refine the keywords and the search string iteratively. This process was validated by using a quasi-gold standard [192]. Finally, the main keywords used for the search are: *Robot*, *End-User Development* and

ID	Description
IC 1	The focus of the article is to describe a EUD or EUP tool for robotics
IC 2	The presented tool is focused to support end users and not expert developers or people working on industrial tasks/robots

Table 5.5: Definitions of Inclusion Criteria (IC)

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

Table 5.6: Definitions of Exclusion Criteria (EC)

Visual Programming. A correlated keyword for *End-User Development* is *End-User Programming*, and one for *Visual Programming* is *Visual Language*. The search string was defined using the Boolean operators as follows: *'Robot' AND ('End-User Development' OR 'End-User Programming' OR 'Visual Programming' OR 'Visual Language')*.

5.4.3 Selection of Papers

The next step in the review protocol is a clear definition of the criterion used to decide which papers are used in this review, and how and when the criteria are applied. The inclusion (IC) and exclusion (EC) criteria for this study are shown in Tables 5.5 and 5.6, respectively.

The following steps indicate how and when the defined inclusion and exclusion criteria are applied: (1) reading the title, abstract and keywords of all articles applying inclusion criteria IC1 and IC2; (2) reading the introduction, contributions, and conclusion of studies included in Step 1 to eliminate irrelevant documents that meet some of the exclusion criteria EC 1, EC 2, EC 3 and EC4; (3) a complete reading of the remaining studies in Step 2 to validate their relevance; and (4) collect all the useful information for the proposed research questions. The search process performed is graphically illustrated in Figure 5.1. As shown in this figure, a total of 1010 articles were returned by an automatic search in the selected databases. From these, 54 were selected after executing step (1). In step (2), after performing skim reading 33 were excluded. Finally, 21 articles were selected for this review. However, some articles such as [193, 26], reference the same interface in different development steps. Therefore in step (3) I identify more relevant and complete articles describing these VPEs. Finally, a total of 16 interfaces were selected for this review. However, I went through all 21 articles for performing data collection.

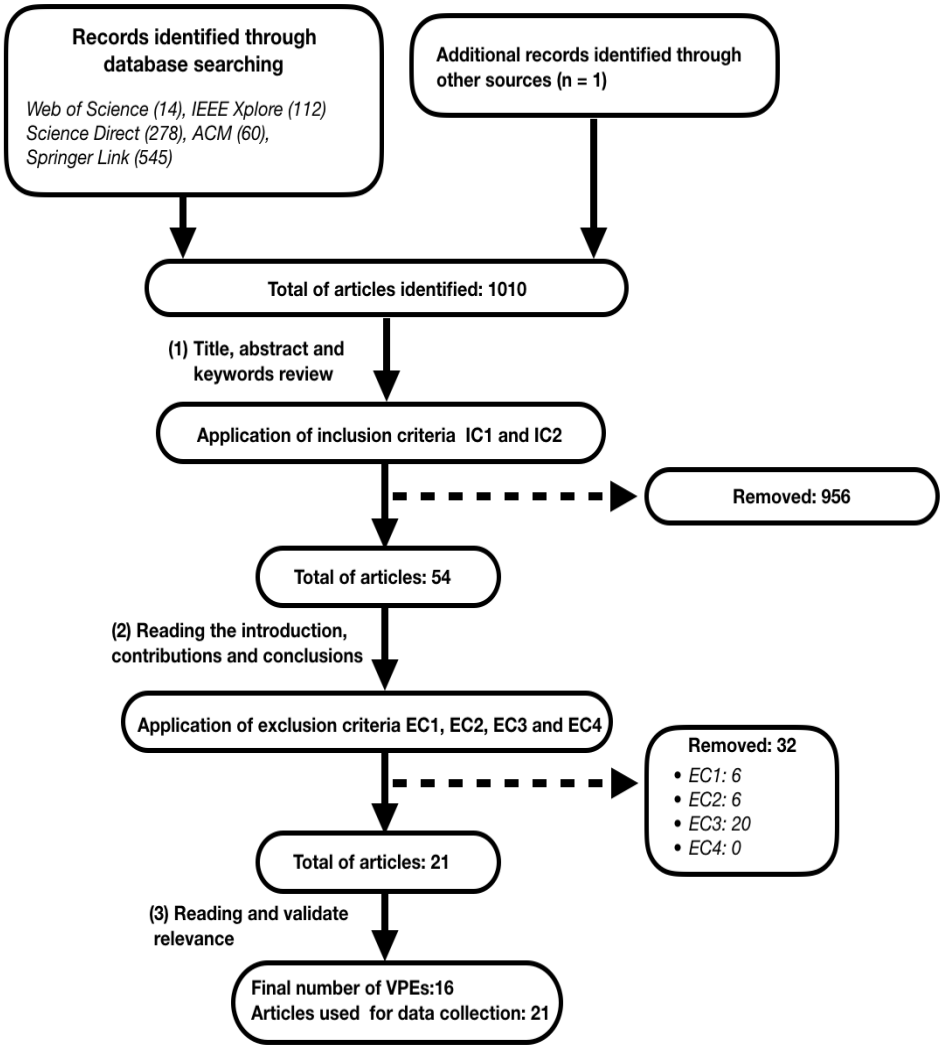


Figure 5.1: Flowchart of the search strategy.

5.4.4 Limitation of the study

The validity of the review may be threatened by three factors, which are described in [194, 195]:

Publication bias is described in [194] as the problem that "positive results are more likely to be published than negative results". In this review, only a few papers of those finally selected report negative results. However, the interpretation of positive or negative results will often depend on the point of view of each researcher [194]. A standard method used to deal with this issue is scanning the gray literature (i.e., master and Ph.D. thesis, books, workshops proceedings, and technical reports). However, there still exists a risk that the presented analysis in this article does not offer a complete overview of the VPEs reviewed.

Interpretive validity is achieved when defined conclusions are reasonable given the extracted data [195]. For this, three researchers experienced in areas such as Software Architectures, Artificial Intelligence, Social Robotics, Usability Engineering were involved in the validation of conclusions.

Theoretical validity is determined by the ability of researchers to capture the intended data [195]. The search process was conducted by an individual author, which is the main threat to validity. Therefore, during the inclusion and exclusion phase, there is a threat some VPEs might have been missed. To reduce this threat I ask experts if they know of any unpublished results or other relevant sources not initially considered in this review. During Data extraction analysis and classification phases, researcher bias is also a threat. To reduce the bias, the three reviewers assessed all extractions made by the one reviewer, such as suggested [195, 196] and applied in [195]. However and described in [195], this threat cannot be eliminated as involves human judgment.

5.4.5 Reporting of results

Next, I answer the research questions of this study based on the dimensions presented. Research questions RQ2 is answered in section 5.5 by presenting a brief overview of the VPEs tools for Social Robotics found after following the proposed search protocol. Section 5.6 is used to answer the research question RQ3 by discussing AAI tools found in the resulting articles. Research question RQ4 is addressed in section 5.7 by presenting and analysing the software tools used in the development of VPEs for social robotics. Finally, research question RQ5 is answered in section 5.8, which presents the identified open challenges.

5.5 VPEs for Social Robotics (RQ2)

To answer RQ2, this section presents a brief description of the VPEs resulting from the systematic search and analysis. I classify these VPEs in three categories: *dataflow-based*, *block-based* and *form-filling*. This classification was explained in section 5.3. Table 5.7 shows the general features and the targets of these VPEs.

5.5.1 Dataflow-based Interfaces

The Microsoft Robotics Developer Studio (MRDS) [197] provides a VPE oriented to enable novice and expert programmers to generate robot-based applications in Microsoft Windows. MRDS is based on C#, includes a 3D simulator and allows for distributed messaging between different modules using a SOAP-based application layer protocol referred to as Decentralized Software Services Protocol (DSSP). MRDS can be used with a set of commercially available robots, including Nao and Kondo KHR-1 humanoid robots. However, the support for MRDS has been discontinued recently.

Name	Target users	Application domain	Robots
CodeIt! [198]	novice and expert programmers	service robots	Sovioke Relay, Turtlebot
OpenRoberta [199]	children, teens	education	Nao, toys
Robokol [200]	therapists	robot-based therapy	Ono
BEESM [201]	novice and expert programmers	smart environments	Turtlebot
RIZE [202]	UX/UI designers	long-term and social HRI, child-robot interaction, entertainment	many
ProCRob [203]	teachers, therapists	robot-based therapy, tutoring	QR
MRDS [197]	novice and expert programmers	autonomous vehicles, competitions, entertainment	many
Choregraphe [190]	novice and expert programmers	social HRI, entertainment, robot-based therapy	Nao, Pepper, Romeo
TiViPE [74]	therapists	robot-based therapy	Nao
Interaction Composer [193]	UX/UI designers	shopping malls	many
RoboStudio [204]	novice and expert programmers	healthcare	iRobiQ-S
RRP-VPE [205]	novice and expert programmers	N.A	Nao
RoVer [206]	UX/UI designers	social HRI	Nao
Interaction Blocks [207]	UX/UI designers	N.A	Nao
English2NAO [208]	therapists	robot-based therapy	Nao
PersRobIoTE [171]	novice programmers	smart environments	Pepper

Table 5.7: General features of VPEs for Social Robotics

Choregraphe [190] is a cross-platform and desktop-based VPE developed by Aldebaran Robotics (now Softbanks Robotics). Its main principle is based on using different wires or connectors to organize multiple robot behaviors in sequence or for parallel execution. The application includes a 3D simulator and allows for designing and debugging robot animations using a time-line interface. Furthermore, it allows designers to develop low-level, Python 2 based scripts, as well as the creation of high-level modules (denoted as *boxes*), which can be saved as libraries for later reuse. The editing of each box parameters can be done using form-based visual interfaces.

The Tino's Visual Programming Environment (TiViPE) [74] is a desktop and data-flow interface built on QT [209]. Originally, it was designed to enable rapid prototyping of robot behaviors using a massively parallel processing and cross-platform approach. In TiViPE, modules can be developed using different programming languages, and can be integrated with their documentation in a stand-alone executable, each one characterized by its own graphical front-end. The development of abstract and complex modules in TiViPE (i.e., made up of simpler, basic, modules) can be done using a form-based interface to combine selected modules. Then, such modules can be reused in the same or other TiViPE-based programs. The only robot supported by TiViPE is Nao. However, unlike Choregraphe, TiViPE allows for the use of multiple Nao robots at the same time specifying their IP address. In TiViPE, the overall robot behavior's control flow is organized using *one-to-many* connections of module input/output ports, i.e., one output port of a module can be connected to more than one input ports of different modules. Optional ports in TiViPE-based modules can be defined to specify relevant parameters needed for subsequent execution. The latest available version of TiViPE enables also the development of sensory-driven dynamic and parallel behaviors, which can be defined using a domain-specific control language. The main real-world use cases in which TiViPE is employed are related to robot-based social therapy.

Interaction Composer [193] is a flowchart-like and interaction-oriented design framework specifically aimed at facilitating the development of social robot applications via coordinating cross-disciplinary teams of expert developers and UX/UI designers, i.e., end users and researchers in social sciences. As envisaged by the original proposers, it is necessary to clearly separate the role of different professional participants in the team. Expert software developers are in charge of low-level programming activities, such as data processing, interfacing

with hardware equipment, and the development of basic robot behaviors in C++, whereas interaction designers only focus on defining the interaction workflow and dialogue generation. Interaction Composer is characterized by a 4-layer, modular architecture enabling the use of different robots sharing a number of similar features and capabilities. Besides the standard interaction workflow, this framework allows for specifying interruptions when certain conditions are met. When an interruption is handled, the control flow resumes from the point where the interaction workflow was interrupted. Furthermore, Interaction Composer allows encapsulating visual elements in a hierarchical way. However, notwithstanding this hierarchical approach, the authors recognize a number of issues related to scalability and flexibility in their dataflow-based interface [26]. The framework has been widely adopted for developing social robot applications in real-world settings, such as shopping malls and supermarkets, and also in situations where a robot acts semi-autonomously [26]. The public availability of Interaction Composer has been discontinued since 2011.

RoboStudio [204] is a desktop-based VPE aimed at the design and development of service robots for medical and healthcare applications. Built on top of the Healthbots framework [210], RoboStudio has been developed using Java and Netbeans, which enable cross-platform support and a low memory footprint. It allows for using software components originally developed for ROS [211, 67], and OpenRTM [210]. As a consequence, RoboStudio is characterized by a high flexibility for the integration of robots and distributed sensors. The design of the application workflow is based on concepts borrowed from Finite State Machines (FSMs). As a consequence, the main programming interface has been designed for expert developers rather than for novices. However, the interface design can be reused and extended to embed other VPEs. A simple example of this is discussed in [204], where a novel interface, called LTLCreator, has been developed on top of modules originally developed using RoboStudio and Netbeans. Algorithms developed using RoboStudio can be converted to an XML-based domain-specific language called Robot Behavior Description Language (RBDL), which can be executed using the Healthbots execution engine [212]. Unfortunately, RoboStudio is not available for use to date.

The Reactive Robot Programming - Visual Programming Environment (RRP-VPE) [205] is a dataflow- and web-based interface powered by Node.js [213]. RRP-VPE is based on the *Reactive Robot Programming* paradigm, which can be described as a "declarative programming approach towards the development of event-driven applications built around the notion of continuous time-varying of data streams and the propagation of change" [214]. Unlike most VPEs based on component-based software engineering approaches, RRP-VPE advocates an approach in which modules can be developed in the same environment rather than using a separate, independent tool. RRP-VPE is based on the notion of *RRP graphs*. These graphs define processes aimed at transforming inputs into outputs using a set of different connectors organized in a possibly complex structure. Examples of such connectors include operators to map certain data to given functions, filter inputs, sample data, or merge different data sources according to a given logic. However, in order to use RRP-VPE users are required to be quite skillful in low-level software development to fully understand the notation related to the declaration of variables, data assignment, and the development of function calls. It is not clear whether non technical end users can adopt RRP-VPEs or not in spite of these low-level notations, and what are the usability and cognitive issues implied by such adoption. RRP-VPE is available online open source [215].

RoVer [206] is an authoring VPE designed with a two-fold purpose in mind: firstly, to enable prototyping of human-robot interaction scenarios built on top of a number of available interaction primitives, and, secondly, to encode appropriate social norms, possibly not known to the designers *a priori* but emerging during the interaction process. RoVer adopts formal verification techniques to ensure that developed programs satisfy a set of social norms en-

coded as logical rules. To this end, RoVer employs the Prism Model Checker [216]. Moreover, this framework is able to provide designers with feedback when a certain social norm cannot be met. Analogously to other frameworks aimed at human-robot (social) interaction, RoVer adopts small behavioral primitives, called *microinteractions*, which can be aggregated to work sequentially or in parallel. Microinteractions can be aggregated in groups, organized as a set of states. Then, the overall human-robot interaction unfolds using a structural, FSM-based architecture, in which transitions between groups of microinteractions depend on the current robot beliefs. RoVer is implemented in Java and can work in Linux or OSX operating systems. Currently, it has been used with the Nao robot. RoVer is available online [217].

Interaction Blocks [207] is a visual authoring environment aimed at the fast prototyping of human-robot interaction processes using Nao. The application uses a set of predefined interaction patterns as basic building blocks to generate more complex interactive processes also sequenced in a time-line fashion. These patterns have been selected by observing different human-human interactions in typical, social settings, e.g., conversations, collaborations, instructions, interviews or storytelling. The main capability exhibited by Interaction Blocks is an easy integration between human-robot interaction patterns and text-to-speech, speech recognition and appropriate gaze behaviors for robots. However, this tool is not available online. The original authors of Interaction Blocks have considered the lessons learned during its development for the design of RoVer [217].

5.5.2 Block-based Interfaces

The Programming Cognitive Robot (ProCRob) environment [203] is a full-fledged software architecture designed to support the development and customization of applications in which social robots are used by teachers and therapists. The architecture has been applied mainly to support innovative therapies for children suffering from the Autistic Spectrum Disorder (ASD) by means of a ROS-based humanoid robot called QT. The ProCRob's architecture is composed of three layers: the first is a functional layer implemented in ROS or YARP [85] made up of software components enabling such basic social skills as gesture expression, text-to-speech, as well as speech, face and object recognition; the second is a middleware embedding a domain-specific language called Robot Agent Programming Language (RobAPL), which uses a Prolog-style rule- and logic-based approach to define goal-oriented behaviors using high-level abstractions and the *Belief-Desire-Intention* (BDI) model [218]; the third is a front-end VPE based on Google Blockly. ProCRob allows its users to represent and manage robot plans based on a set of tasks organized sequentially or in parallel on the basis of *a priori* commands or external events. The basic workflow unit is called *play*, which is represented by a behavioral block embedding text, audio, face expressions, and body animations. However, ProCRob is not available online.

CustomPrograms/Codeit! [198] is a Google Blockly and web-based interface designed to reproduce the expressiveness of general-purpose programming languages by the use of low-level constructs such as loops, variables, math utilities and functions. Built on top of Node.js and roslibjs [75], it also provides a set of high-level programming abstractions denoted as *primitives*. However, it is explicitly mentioned in [219] that the use of general-purpose programming language constructs, while they can be easily and intuitively used by experienced programmers, require more training and generate more complex systems when used by inexperienced programmers. CustomPrograms/CodeIt! has been used for a series of end-user applications with mobile service robots in exhibitions, hotels [219], and for STEM-based training programs [220]. It is noteworthy that one of the main advantages of this interface is its compatibility with ROS-based software modules. However, one of its strongest drawbacks is the impossibility of reusing code due to the limitations of the default features of the Google Blockly library. A

study evaluating the ease-of-use and expressiveness of CustomPrograms/CodeIt! is reported in [219]. This framework is available open source [198].

OpenRoberta [199] is a block-based VPE mainly used for educational aims. However, this VPE has also been used in end-user applications [221]. OpenRoberta is based on Google Blockly, and enables software development for a variety of toy robots, single-board micro-controllers, and the social robot Nao. Unlike most VPEs analyzed in this article, OpenRoberta comes in two versions. In the first version, it can be run as a browser app connected to the Internet using a cloud-based server as a back-end. This option simplifies to a great extent installation and setup. In the second version, it is based on an offline, Java-based and cross-platform local server. However, due to its mainly educational-oriented target, OpenRoberta exploits many low-level development abstraction primitives, which must be grounded to the use of classical programming abstractions. This approach is in fact more suitable for educational purposes.

Robokol [200] is oriented to non-programmers, and is focused particularly on the development of applications in support for ASD-related therapy. The Robokol's interface is powered by Snap! [180, 222], and enables cross-platform support. Such support is possible by connecting external devices to a data exchange server (e.g., a remote computer running ROS). This connection can be established by a plug-and-play approach (i.e., the device advertises its own description rather than requiring a user-specific setup) via websockets using the ROSbridge protocol and suite [75]. Experimental settings where Robokol has been adopted are related to the use of the Ono social robot, as well as a therapeutic device called the *Sensory Sleeve* [200]. Like CodeIt!, Robokol uses general-purpose programming language abstractions. The framework does not seem to be available online.

The Block-based End-user programming tool for SMart Environments (BEESM) [201] is a VPE framework based on Google Blockly. The application allows for rapid prototyping of applications involving smart environments, microcontrollers and mobile robots. Like Robokol and CodeIt!, the back-end is based on ROS, and the whole framework mainly adopts low-level general-purpose programming notations. This low-level abstraction enables the users to learn PHP and how to code with Arduino boards, which is required to program smart environments and mobile robots with the supported middlewares and libraries of this interface. It also includes a 2D simulator for smart environments and mobile robots. It is reported that the BEESM interface will be evaluated in usability tests soon. However, BEESM is not available online yet.

The Robot Interfaces from Zero Experience (RIZE) framework [202] is a cross-platform, block-, form- and web-based interface enabling remote control and the generation of intelligent authoring behaviors for different robots. This VPE is one of the main outcomes of this thesis. Therefore, a deep description of this VPE is performed in chapter 6. Unlike a majority of block-based interfaces based on Google Blockly, RIZE does not adopt general-purpose software development abstractions. On the contrary, it uses a modular approach based on the definition of independent behaviors that can be easily reused in other RIZE-based programs. Robot behaviors are encoded as *behavior trees*, i.e., a meta-architecture for the generation of reactive, modular, and complex agents [147], and are executed by a decision-making engine. RIZE has been used for the remote control and the generation of intelligent behaviors using a ROS-based Turtlebot Burger robot, Nao and Pepper humanoid robots, as well as a robot manipulator built with Dynamixel servomotors and controlled in Matlab/Simulink. Real-world applications include museum exhibitions and theater performance [21], child-robot interaction [20, 223], long-term human-robot interaction experiments in home settings and research in emotional intelligence for robots [224]. RIZE is available online [202].

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

Table 5.8: Comparison between AAI approaches using in VPEs (RQ3)

5.5.3 Form-filling Interfaces

English2NAO [208] is an EUP tool in which programming inputs can be set both by natural language processing and with a form-filling interface for enabling the therapists to create programs for NAO robot. This interface is developed as a web-based application using Django [225], and runs on top of the TiViPE engine. This EUP tool was developed to overcome some of the usability problems presented by TiViPE [208]. Online availability of this English2NAO has not been reported.

PersRobIoTE [171] is a web-based, form-filling interface. It adapts the Trigger-Action Programming (often used in EUD tools for IoT scenarios) paradigm for enabling the creation applications with Pepper robot. Users of PersRobIoTE need to define a set of rules mainly composed of triggers (conditions concatenated by and/or Boolean operators) and actions. These rules are encoded in the JavaScript Object Notation (JSON), which are created and managed by a decision-maker module called the Rule Manager. Moreover, it uses backboard-like modules, referred to as their authors, such as Context Manager to handle perceptual inputs from both Pepper robot and IoT devices. Communication between these modules is performed using the Server Sent Events (SSE) [226]. However, PersRobIoTE is not available online yet.

5.6 Modeling Intelligent Behaviors (RQ3)

In this section, research question RQ3 was addressed. RQ3 aims to discover and analyze the AAI tools used for supporting EUP and EUD in Social Robotics. RQ3 is addressed in three ways: (i) presenting a general description as well as advantages and drawbacks of those AAI tools for supporting VPEs; (ii) analyzing the modular capabilities of these AAI tools and (iii) identifying the abstraction levels (programming primitives) generally used in these approaches.. The values of dimensions used in RQ3 are shown in Table 5.8. This table also includes the dimension RQ2-D2 (EUP approach) for comparative purposes.

5.6.1 Scripting-based

As shown in Table 7, most block-based programming VPEs, such as Open-Roberta, Robokol, and CodeIt! use general purpose scripting to enable the creation of applications with social robots. In this approach, end users need to become familiar with classical scripting approaches

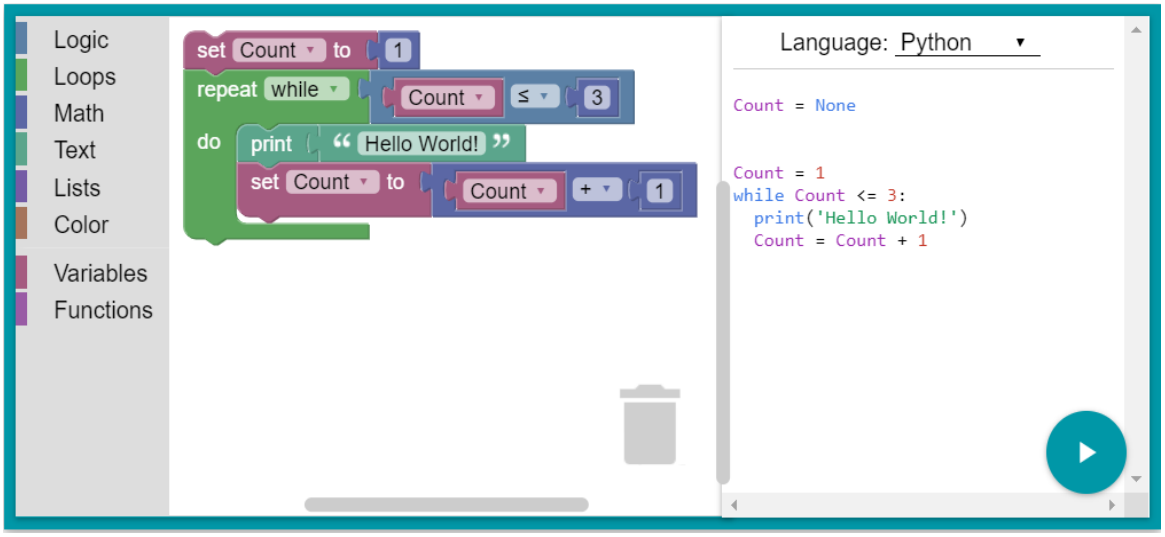


Figure 5.2: Example of a block-based programming environment using general purpose programming notations using Google Blockly, end-user code is converted to real code in some programming language

such as if-else conditional statements, for loops, creating variables and functions, and using low-level mathematical and logical operations. The acquisition of these low-level programming skills is a major objective of STEM educational courses. However, the suitability of this approach for enabling the creation of intelligent robots by end users can be hindered by usability and code reusability issues. As described in [156], the Cognitive Dimension framework suggests that using too many low-level programming notations may produce usability problems associated with high viscosity (i.e., users need to manipulate many elements to accomplish a task), and may require hard mental operations and distant mapping to the problem domain of social interaction. Moreover, using general descriptions for programming agents may produce code that is hard to reuse and maintain [147].

5.6.2 Rule-based

First computer games and robots used programming and modeling systems using rules (e.g., if-this-then-that) for building intelligent agents. This approach is simple to implement and presents a uniform method of representation, which can be intuitively used by non-programmers [227]. However, these systems also present many relevant drawbacks. While non-programmers can easily grasp the approach of using individual rules for programming robots, they face difficulty in going beyond the declarative approach based on using rules. They also have difficulty in understanding the implications of multiple rules, some of which may conflict. Moreover, the lack of structure in rule-based systems often leads to (i) maintainability issues and error-prone handling of programming elements (rules) in complex systems [148], and (ii) unstable and unexpected behaviors when creating very large and complex programs [3]. While programming approaches using disjoint and priority-based rules are currently very popular for developing IoT applications, they have been widely replaced by more structured and robust AI architectures, such as Finite State Machines and Behavior Trees, in areas requiring a development of more complex, social and interactive agents, such as game developing and robotics. As shown in Table 5.8, VPEs using rules as the main modeling tool are generally developed using form-filling interfaces. The approach used on PersRoboIoTE is similar to EUD solutions for IoT systems, such as IFTTT [228], where a list of disjoint rules must be filled. Figure 5.3 shown a simple

Figure 5.3: Simplest notation used in a rule-based VPEs

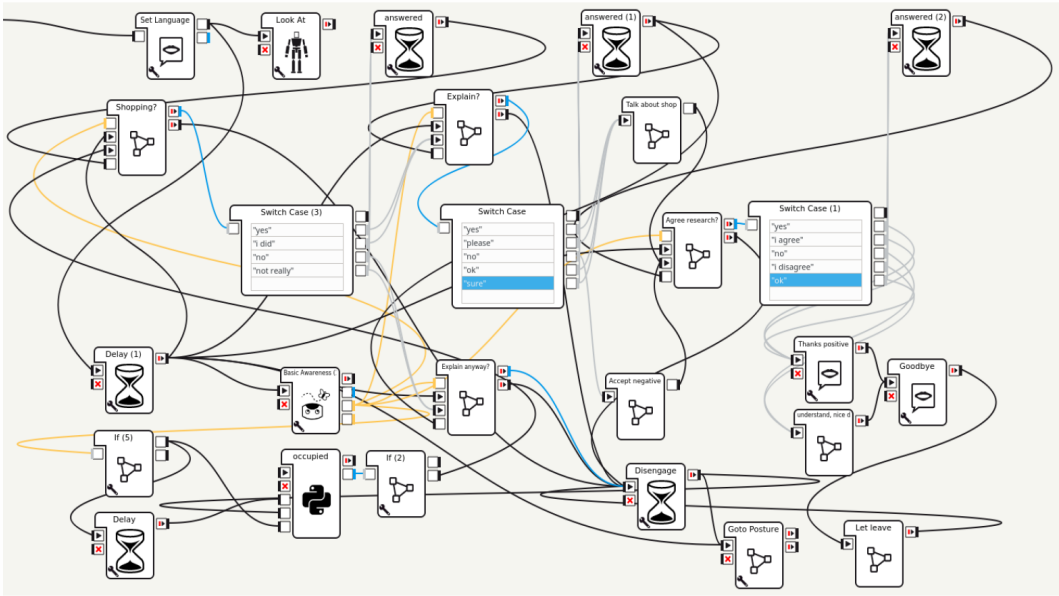


Figure 5.4: Example of spaghetti code in a dataflow-based VPE (example taken from [2].

visual notation for systems using rule-based systems and form-filling. For these VPEs, the main programming task of end users is to select a condition or set of conditions (concatenated with logical operators AND/OR) that will trigger some robot action.

5.6.3 State-based

By including the notations of states and transitions (i.e., some decision logic that told the system to change from one state to other), as well as adding a structure to a set of disjointed rules, a rule-based system turns into a state-based method. Most popular method used to model state-driven systems are Finite State Machines (FSMs). FSMs are represented as directed graphs, where each node of the graph represents a state. In an FSM, each transition to a new state represents an event. These events can trigger the execution of some specific script or sequence of robot actions. In general, FSMs are robust and easy to understand even for novice end users [229]. However, the use of FSMs inherently implies some reactivity and modularity issues, which are analogous to those associated with *goto* statement [148]. Both the *goto* statement and FSM can be considered as a "one-way" control" transfer (i.e., the control flow jumps to another section of the program), which is described in [148, 230] as "too much an invitation to make a mess of one's program". As described in [148], this leads to a trade-off between reactivity and modularity of the programming system. To create a reactive and complex social interactive application, a program built using FSMs requires too many one-way control transitions between visual elements. As shown in figure 5.4, this requires the creation of very tangled diagrams where the modification or removal of some element may need checking every transition and state associated with that component.

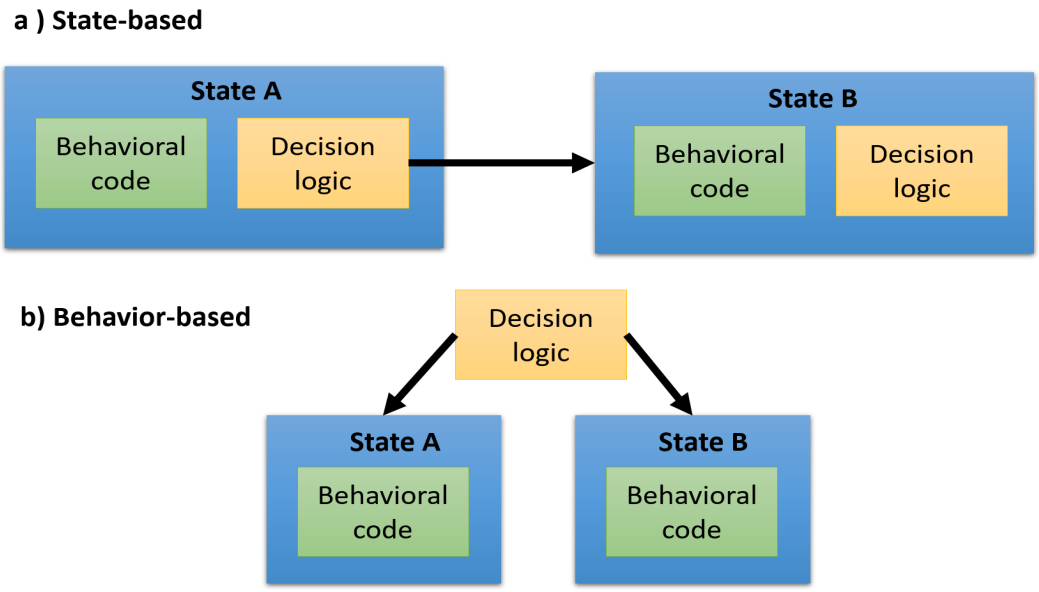


Figure 5.5: a) In state-based methods, each state requires the definition of the decision logic that indicates the decision-making system how to change to another specific state; b) Behavior-based approaches separate decision logic from behavior code enabling a hierarchical and modular representation (adapted from [3])

5.6.4 Behavior-based

By hierarchically organizing and separating the decision logic from the behavior code, a state-based system turns into a behavior-based approach. Figure 5.5 shows the main difference between state-based and behavior-based modeling methods.

As proved in [174, 148] most behavior-based modeling methods used in robotics can be generalized by a Behavior Tree (BT). Unlike FSM, BT is considered a "two-way" control transfer — after the execution of an event or function, the control flow returns to the part of the program in which is called — which enhances modularity [148]. A typical BT implementation is composed of two types of nodes: operators and terminal nodes. Figure 5.6 shows an example of a simple BT. While operator nodes (in white) are used to perform control flow and behavior selection, terminal nodes (in blue and gray) define and check preconditions and execute the proper agent behaviors. More basic operators in a BTs are *Sequence* and *Selector*. Functionality of these and other common operators in BTs are described in [148]. The execution of a BT follows a classical "depth-first" traversal order from the root node to some terminal node. After the activation of a node (when the BT traversal algorithm reached the node), this node is assigned a status. This status can be "success", "failure" or "running" depending on the type of node. Each iteration of the BT traversal algorithm performs decision-making tasks depending on the status of these nodes. BTs are also, by definition, modular and reusable [174] as each branch of a BT can be considered as an independent module. Figure 5.6 shows four possible modules that can be easily reused in other programs. Unlike FSMs, BTs have just started to gain attention in robotics. Therefore, available software frameworks supporting this AAI approach are less mature [21, 148]. Moreover, concepts involved in the creation of BT can be difficult to understand by end users, as it is required to learn the "depth-first" traversal graph search and many low-level operators for performing decision-making. An alternative aimed to enable the use of BTs for end users is proposed in RIZE by changing the way BTs are modeled. In the approach proposed in RIZE, rather than allowing the end user to build and execute BTs using a tree structure and low-level operators, the end user build their programs

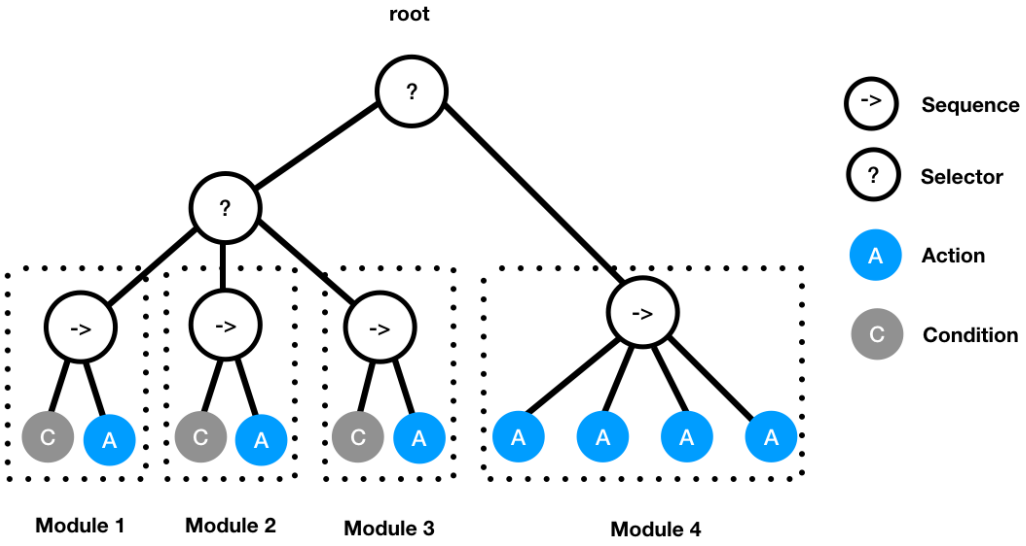


Figure 5.6: Example of behavior tree

by concatenating a set of BT modules or sub-BT in a declarative way using a Google Blocky environment. These high-level modules or social primitives are built by expert programmers using a low-level domain-specific language. More details are described in [21].

5.7 Tools, technologies and evaluation methods used in VPEs for Social Robotics (RQ4)

As shown in Table A.1, Web technologies, such as HTML and Node.js (Javascript), are preferred to build VPEs for Social Robotics. However, only OpenRoberta is built as a Web service. Instead, other VPEs using Web technologies are designed to be executed on a desktop using server-side frameworks such as Node.js and Django. For building block-based programming environments, the preferred tool is Google Blockly [181], which provides more features and flexibility than similar tools such as Snap [180] and Scratch [179]. As was described in Sections 5.3 and 5.6, some block-based VPEs using Google Blockly (specifically RIZE and ProCBob) use this tool as a domain-specific language, where the code is executed in a more advanced AI architecture (BTs in the case of RIZE and BDI in the case of ProCBob), rather than in a general purpose programming tool. Moreover, older VPEs, such as Choregraphe, TiViPe, and MRD were built as *desktop-based* tool for developing user interfaces such as Visual Studio and Qt. The only recent VPE reported to be built as a classical *desktop-based* interface is Rover, which was implemented in Java version 8. Table A.1 shows that many recent VPEs use some Component-Based Software Engineering (CBSE) frameworks, with ROS. In this approach, software modules are seen as isolated processes or nodes that are executed in parallel and that can be written in different programming languages. This approach enables an easy reuse of many open-source software tools developed by the robotics community thereby creating more robust and complex robotic systems. However, most of these recent VPEs, such as CodeIt!, Robokol and BEESM, require the execution of a server module or node in a computer with the right version of Ubuntu installed. This can be a barrier to their adoption by the end users for use-time design activities. This is due to a steep learning curve of ROS frameworks [24]. Drawbacks of ROS for EUD were described by the creators of TiViPE in [73] as: (i) most of

the end users are Windows users and require easy-to-install software tools; and (ii) they hardly understand (without training) many of the concepts required to use ROS. Therefore, a high-tech scribe skilled in the ROS framework is often required for the installation and execution of the web server and for maintaining or extending these VPEs, which can be performed by launching additional ROS nodes. These issues are still not solved with ROS 2.0, as it requires following complex steps for its installation in Windows 10 and training for their use. Due to these issues, interfaces such as TiViPE, MRD, and Choregraphe have been developed as monolithic applications that are easy to install for the average end user , and where software modules are executed in different threads. This can explain why only VPEs that have easy-to-use wizard installers have been reported as enabling both the creation of applications at design time and their redesign at use-time by the real end users. To communicate with external modules, VPEs build as monolithic applications generally use simple POSIX sockets for some limited tasks. From Table A.1, it is possible to see that most popular method used to validate the suitability of these VPEs is the System Usability Scale (SUS) [231], which allows a reliable and valid evaluation of usability for a wide variety of products and services such as hardware, software, websites, and applications. Another method used to detect usability problems is the Cognitive Dimension Framework, which was described in Section 5.2. The NASA Task Load Index (NASA-TLX) and Cyclomatic complexity metric have been also been used to measure perceived workload and complexity when creating programs for social robots. However, these evaluation methods are often used to obtain subjective data. In the case of Codeit!, objective and quantitative evaluations, such as the time and number of successful tasks, have been performed.

5.8 Open Challenges of EUD for Social Robotics (RQ5)

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

5.8.1 Accessibility to External Devices and Resources

This open challenge is based on the data obtained from dimensions RQ4-D3 (Accessibility), RQ4-D4 (Operating Systems), and RQ4-D5 (Easy-to-install and execute). An analysis of these dimensions reveals a number of accessibility issues in recent VPEs analyzed. From the point of view of non-technical end users, such as UX/UI designers, an accessible VPE requires (i) accessibility for their use or evaluation (RQ4-D3), (ii) compatibility with end-users devices (RQ4-D4), and (iii) user-friendly installation and configuration (RQ4-D5).

Accessibility for their use or evaluation . As shown by the values obtained in dimension RQ4-D3 (Accessibility) Table A.1, more than half of the presented VPEs are not available online, even though many of them are built with open source tools. This hinders a proper evaluation of them and denies opportunities to obtain feedback from both the robotics community and end users.

Compatibility with end-user devices. In many professional environments and in the Academia Linux-based systems have a good reputation and impact. However, end users who are not technically trained often assume the availability of software designed and developed for Microsoft Windows or OSX, which have a bigger market share compared to Linux-based systems for the general consumer market. While this issue is well-known and deeply understood by most

commercial VPEs, such as Choregraphe and TiViPE, it is mainly ignored by most community or open source VPEs, therefore jeopardizing their ability to gain widespread use. A common argument used to justify cross-platform support is to design the VPE's architecture as a (possibly web-based) front-end running on Microsoft Windows or OSX coupled with a back-end typically running on a Linux-based system, which is done by Robokol and the offline version of OpenRoberta. However, such an architecture still requires the server side to be configured on a Linux-based platform.

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

5.8.2 Modularity of Human-Robot Interaction Primitives

In Computer Science, and in particular, as far as a software architecture for robots is concerned, the word *modularity* is ambiguous and can be related to concepts present at different levels of the architecture and granularity scales. For this work, I consider two different meanings associated with the notion of modularity, namely *operational* and *structural* modularity. The formulation of this open challenge is based on the data obtained from dimension RQ3-D1 (Communication) in Table A.1, which is related with *operational* modularity, and the analysis presented in Section 5.6 about modular and reusable capabilities of each AAI method used for modeling social robot behaviors, which is related to *structural* modularity.

Operational modularity. While a few authors consider modularity in VPEs as a simple encapsulation of function calls or a set of related functions, others aim at integrating higher-level modular abstractions, such as the *Separation of Concerns* design principle in independent processes (also denoted as nodes) and/or software packages, as it is done for instance in ROS [67]. Recently, the latter approach has been the most successful, being considered as the best practice for Robotics-related software development [232], and one that provides an increased quality in software applications. According to this approach, and as far as VPEs for Robotics-related applications are concerned, data exchange between processes is managed on the basis of a number of well-defined inter-process communication patterns [233].

Based on these concepts, and the values of dimension RQ4-D1 in Table A.1, it is possible to classify VPE interfaces as having low, tight, or high operational modularity. In the first case, modules are just considered as a set of function calls. In this approach, most of the robot's sensory, perceptual, decision-making and control tasks are carried out as parts of a single process. VPEs that exhibit low operational modularity include Interaction Blocks and

RoVer. In the second case, the overall robot functionality is split in different modules, and various modules communicate with each other using a *Request-Process-Reply* (or *Client/Server*) design pattern [234] through POSIX sockets. VPEs adopting this operational modularity type are MRDS, Choregraphe, TiViPE, Interaction Composer, and OpenRoberta. In the third case (also referred to as *loose coupling*), the principles of reusability, extensibility, maintainability, and robustness are enforced by the use of non-blocking and asynchronous communication design patterns, such as *Publish/Subscribe* and *Observer*. VPEs in this class are RRP-VPE, RobotStudio, ProCRob, CustomPrograms/CodeIt!, Robokol, BEESM, RIZE.

Structural modularity. One of the main drawbacks associated with most of the analyzed VPEs, particularly those based on the flowchart concept of FSMs, is the lack of modularity, intended as a clear subdivision of roles within the resulting architecture. Such a lack of modularity-enforced design is due to the mainstream approach to organize internal workflow (i.e., mostly corresponding to the overall robot decision-making capabilities) as a single, one-way, data transfer from inputs to outputs. In a sense, such approaches replicate the somewhat classical Sense-Plan-Act architecture for robot perception, planning and control [235]. The result is characterized by issues similar to what happens with the use of the much critiqued *goto* statement, which is considered unstructured and a cause of unreliable behavior [236, 148]. As a consequence, dataflow-based VPEs tend to generate *spaghetti* code (Figure 5.4) and visual programs difficult to understand, maintain, reuse, and scale [26]. As described in [237] and reported in [26], VPEs using dataflow present three key issues : (i) their programs tend to be very large requiring the creation of too many nodes even for trivial and low-level tasks; (ii) each node requires too many inputs and links between them producing highly tangled programs (referred to as *spaghetti* code), such as shown in the example in Figure 5.4; and (iii) confusing iteration: the program is difficult to follow or even understand. To deal with these drawbacks, a viable solution could be to develop truly independent and modular systems based on two-way data exchange, e.g., the adoption of block-based interfaces capable of using service-like approaches based on events and handle functions, on rule-based systems and/or hierarchical decision-making engines like the Behavior Trees [238].

5.8.3 Scalability in Large Applications

In most cases, designers and developers of VPE-based authoring tools advertise use cases in which their frameworks are adopted to design simple human-robot social interaction patterns requiring the use of few primitive behavioral blocks and connections. Like many virtual agents [30], the development of more complex or long-term applications to be delivered in everyday scenarios, such as those described in [49, 239, 240], can require the integration of a large number of behaviors, as well as a possibly intricate logic to coordinate their orchestration. This requirement implies a rapid increase in the difficulty in following the application's control flow, and in searching for appropriate primitive robot behaviors. Module or component encapsulation is a widely-adopted approach used in dataflow-based VPEs to deal with these issues. In many cases, however, encapsulation reduces the *mess* of dataflow-based workflow only to a limited extent [148]. Suitable design patterns to deal with these issues are rare in block-based interfaces.

5.8.4 Correct Abstraction Levels and Programming Notations

As analyzed in [156] and according the values obtained in dimension RQ4-D2 (Programming primitives) in Table 5.8, many of the VPE-based frameworks discussed in this article are characterized by unbalanced abstraction levels in selecting robot behavioral primitives and programming notations. In fact, typical issues are related to the presence of primitives with low-level and varying abstraction levels, and to the consequence of the overall VPE usability [153].

As far the abstraction level is concerned VPEs, such VPEs as Choregraphe, OpenRoberta, TiViPE and Interaction Composer are characterized by several issues in usability and in the cognitive dimension, due to the fact that they incorporate various low-level programming abstractions, which are denoted in [156] as hardware and algorithm primitives. On the one hand, VPEs including hardware primitives use graphical elements which enable users to obtain raw data (e.g., position, velocity, sound and images) from sensory devices or actuators. On the other hand, VPEs including algorithm primitives require that users be able to provide the data generated by the hardware primitives as inputs of low-level perceptual and control modules (e.g., sound source localization, inverse kinematics, keyframe animation and face tracking). However, raising the level of programming abstraction too high, as it is done for example in Interaction Blocks, can reduce the flexibility of VPEs and therefore the capability to create complex behaviors with robots. An alternative approach allowing for a good trade-off between the usability and flexibility of VPEs and robot programming software aimed at generating social interactions is also described in [156]. The correct level of abstraction for developing social interaction behaviors with robots requires the use of reusable and atomic domain-specific social primitives (e.g., related to speaking, gestures, gaze, facial expressions and animations).

5.8.5 Benchmarking

The evaluation of interfaces with real end users is a key task that enables to show the applicability of VPEs as well as to obtain valuable data to validate or improve usability. These evaluations require data collection from both objective and subjective approaches. The collection of objective data is based on facts rather than opinions or interpretations (e.g., how many times the user makes an error, the number of times that a user has required help, and task completion time). This type of data is generally collected and analyzed by those VPEs reporting usability evaluations. From the values obtained in dimension RQ4-D7 (Evaluation methods) in Table A.1, it is possible to observe that authors of TiViPe and CodeIt!/CustomPrograms have used, to different degrees, the Cognitive Dimension Framework (CDF) as a main tool to perform subjective data analysis. This framework is always used to identify usability trade-offs in early stages of designs and make decisions about those trade-offs for posterior iterations [241]. While the CDF has emerged as the predominant framework for analyzing VPL, some researchers have identified some serious theoretical and practical limitations of CDF for its use in the evaluation and design of visual notations [242]. Some of the main issues of CDF described in [242] are: (i) confusion or misinterpretation when interpreting and applying dimensions, (ii) lack of evaluation procedures or metrics, (iii) and the omission of issues around whether the visual notations chosen are "good" or "bad" ones. A complementary approach for addressing CDF issues is to follow guidelines and the principles of the Physics of Notation [242], which are valuable tools for evaluating and designing visual notations. However, these guidelines and principles are often omitted in most of the VPEs reviewed. A well-accepted subjective data collection approach in Human-Computer Interaction (HCI) is the use of standard questionnaires [243] such as the System Usability Scale (SUS) [244] and the NASA Task Load Index (TLX) [245]. From the VPEs reviewed in this chapter, only Interaction Blocks (using SUS), RRP-VPE (using TLX), and RoVer (using both SUS and TLX) have performed subjective data collection using standard questionnaires. However, only the designers of Interaction Blocks have reported usability evaluations using real novice end users, rather than expert programmers.

A comparative evaluation among interfaces, using objective and subjective data, is a valuable task in modern HCI research. This evaluation enhances the analysis and validates the suitability of proposed VPEs. However, this task is often omitted by the designers and developers of most VPEs presented in this work. Exceptions using the NAO robot are presented in [205] and [208]. Issues that limit performing such comparative evaluations are related to

the facts that (i) some of the presented VPEs are not available online, and (ii) they support different robots and target user groups. Recently, a shift of emphasis in many areas of HCI to user experience has become a central focus for interfaces design and evaluation [246]. However, most UX-related aspects [247], except usability, have generally been omitted in the reviewed VPEs.

5.8.6 Explainability and Generation of Robot Social Behaviors

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

C_1 repetitive, robot behaviors that do not require any specific form of high-level intelligence nor cognitive capabilities;

C_2 scripted sequences of basic robot behaviors, the execution of which always follows a predefined list of actions and occasionally requires user-provided input, e.g., using keyboards, joysticks, touch or speech, to continue execution;

C_3 state- or event-based *dynamic* behaviors that do not allow for any interruption nor preemption until the current behavior is completed;

C_4 state- or event-based reactive and dynamic behaviors enabling interruption, state change or preemption on the basis of predefined priorities;

C_5 hybrid reactive/deliberative, intelligent behaviors that require the robot to correctly interpret and react to external and internal stimuli, and to make decisions about which actions to perform next in view of a certain goal.

While most of the VPEs reviewed in this paper can generate behaviors belonging to classes C_1, C_2, and C_3, very few are capable of generating robot behaviors which can be classified as C_4 or C_5. An exception that can be considered as a basic approach towards C_4 is Interaction Composer, which exhibits a dataflow interruption mechanism, where an interrupt could monitor some perceptual input and trigger another behavior sequence [193]. Another exception towards C_4 is TiViPE, the latest release of which includes a textual robot language that can be used to set the execution priorities of a set of serial and parallel actions. However, this low-level textual approach mixed with TiViPE has been reported in [208] to be complex for being used by end users. More advanced approaches towards C_4 have been proposed by RRP-VPE and RIZE, using Reactive Programming and Behavior Trees, respectively. However, it is apparent that neither RRP-VPE nor RIZE have been used with non-programmers, while their adoption by expert software developers and academics has been reported. However, it is noteworthy that while moving from C_1 to C_5 in the behavior classification, the resulting robot actions may be considered progressively less understandable by and explainable for humans. This is because the composition of many simple behaviors in an intertwined chain of planned actions and reactions to certain events can lead to widely different outcomes, even when there are only small differences in the (sequences of) inputs [248, 249]. It is not surprising that, as far as human-robot interaction is concerned, social robots exhibiting predictable behaviors are to be preferred given the state-of-the-art knowledge in Robotics.

5.8.7 Simulation and Debugging

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

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

Cross-platform, easy-to-use and easy-to-setup simulators are the key to increase the overall usability of VPE-based development. Values obtained from dimension RQ4-D6 (Liveness and Simulation) in Table A.1, show that most of the VPEs discussed in this article lack any robot behavior and human-robot interaction simulation capabilities, the only exceptions being Choregraphe, which provides a 3D simulation for the robots commercialized by Softbank Robotics, BEESM, which includes a 2D simulator for smart environments, and OpenRoberta, which provide 2D web simulators of toy robots. Values obtained from dimension RQ4-D6 also indicate that less than half VPEs provide on-demand feedback or debugging (liveness 3), and only Choregraphe provides live feedback capabilities (liveness 4). The concept of liveness was discussed in Section 5.4.1 in the definition of research question RQ4.

5.9 Conclusions

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

scripting, and data-flow programming, thereby providing end users with more reliable, reusable and reactive programming tools to enable the creation of more complex behavior for social robots. Unlike other information technology areas, where end-to-end black-box AI architectures are currently trending (e.g. Deep Neural Networks), AI architectures for enabling EUD of social robots mostly focus on the use of AI tool with authoring and explainable behaviors. This situation is similar to the one facing the game developers, where the creation of robust, explainable and suitable behaviors (in many cases defined by UX/UI designers) is more valuable than learning capabilities. However, the creation of more complex robotics systems will require Social Robots to learn from its environment.

Chapter 6

Designing RIZE End-User Development Framework

This chapter presents design considerations and main features of RIZE (Robot Interface from Zero Expertise), a EUD tool oriented to support interdisciplinary HRI research activities. RIZE is mainly a block-, form- and web-based programming environment which enables end-users to intelligent social robots. RIZE was designed to overcome many of the issues of VPEs for robotics, which was presented in chapter 5. For this, the use of NEP as a robotic distributed framework plays a relevant role.

6.1 Usability and UX in HCI

Two key concepts in HCI research are usability and UX. On the one hand, usability goals are generally regarded to ensure that interactive products are easy to learn, easy to remember, effective, efficient, and safe. On the other hand, UX goals are generally regarded as the design of more subjective goals such as fun, enjoyable, pleasurable and aesthetically pleasing interfaces. In order to develop interfaces that provide high usability and UX, researchers in HCI have proposed some design guidelines. One of the most known is described in [250]. However, these guidelines always require to deal with some trade-offs [251], and their correct implementation will depend on the application. Guidelines used for designing RIZE are shown in table 6.1. An important concept used to improve subjective UX is aesthetics. Interface with a careful aesthetics design not only provide satisfaction and pleasure to the users but also can influences in numerous factors, such as a user's first impression, perceived usability and interface performance [252]. Therefore, interfaces developed in this thesis are designed using modern web-based technologies (described in chapter 3) and design approaches (such as flat, minimalism, and material design). This is done with the objective to develop usable, modern, attractive and beautiful user interfaces. This approach highly contrasts with most state-of-art EUP and EUD

Table 6.1: Usability guidelines used for the design of the NEP interface

Guideline	Description
Consistency	Use accepted standards or conventions to avoid confusions
Feedback	Clearly shown the current state of the system and avoid mode errors
Recovery	Users must be easily recognized and recover from errors
Minimal design	Reduce information overload
Simplicity	Create as simple as possible task-oriented interfaces
Aesthetics	Users perceive more usable and valuable aesthetically products that ugly ones
Documentation	Help resources must be task-oriented and well organized

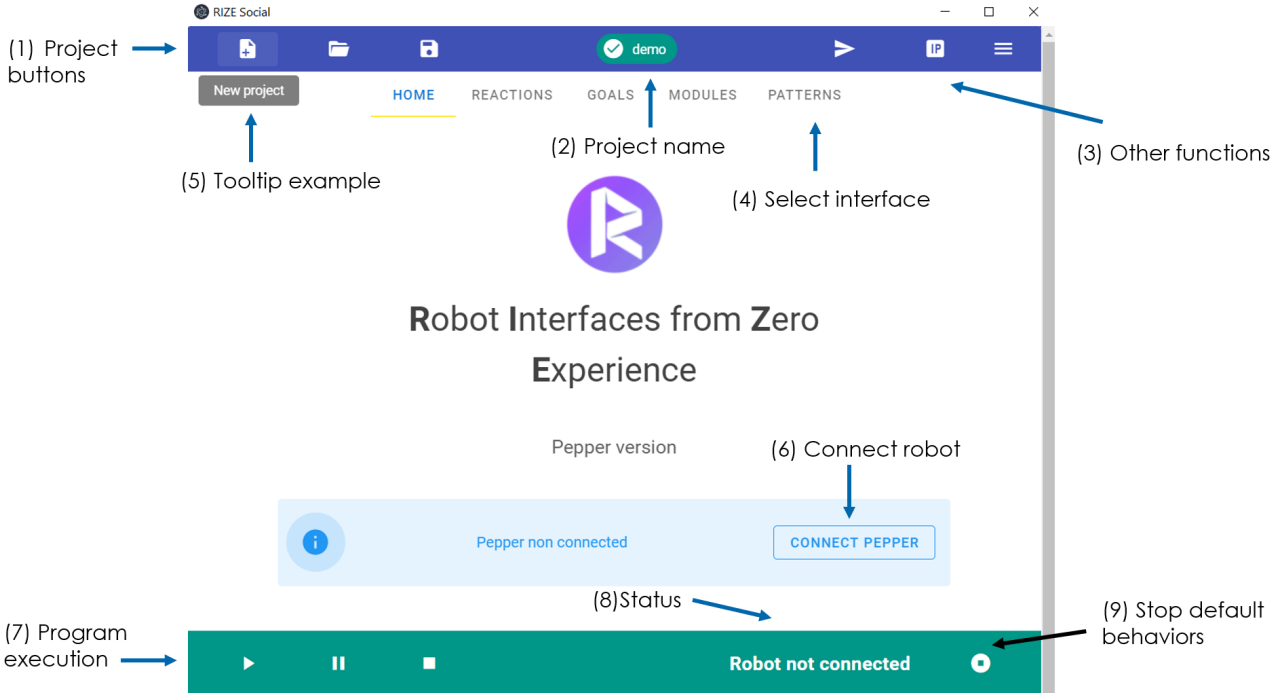


Figure 6.1: Home interface of RIZE

tools for robotics as well as interfaces available in distributed robotics frameworks which are developed using classical tools for build user interfaces and older design paradigms such as skeuomorphism.

6.2 Design

The RIZE interface was oriented to be aesthetically-pleasing in order to convey an aesthetic-usability phenomena [253] (i.e. beautiful design are considered more usable and valuable that ugly ones). Therefore, rather than use classical tools for developing user interfaces in Desktop applications, such as Visual Studio and Qt, RIZE was developed using modern web-based technologies (described in chapter 3). The design of RIZE is based on Vuetify.js [254], which is a material design frameworks based in Vue.js. This open source library enable the creation of intuitive and beautiful digital experiences. As described in [255] material design is group of "guidelines, components, and tools that support the best practices of user interface design". Guidelines of material design were originally proposed by Google, which uses this design approach in all their products. Examples are Gmail, Google Docs, Google Translator, Google Maps, interfaces in Android and Google Search. These are in fact highly used tools and interfaces widely used by the general people in their everyday life. Therefore, the use of this design approach not only enable the creation of more beautiful and modern users interfaces but also more familiar and consistent with the conventions often used by end users. RIZE uses a minimal and flat design in the buttons, programming and other interface elements to reduce the information noise presented in skeuomorphic interfaces. In order to reduce cognitive load and preserve consistency RIZE uses standard icons as well as tooltips that indicates the functionalities of each button. Figure 6.1 shows the home tab of RIZE. Interfaces in RIZE have a top toolbar where users can use a set of buttons on the left side to create new projects, load a project and save the current project (1). This menu shows in the center the name of the project (2). On the right side (3), a set of a button with more advanced features is displayed.

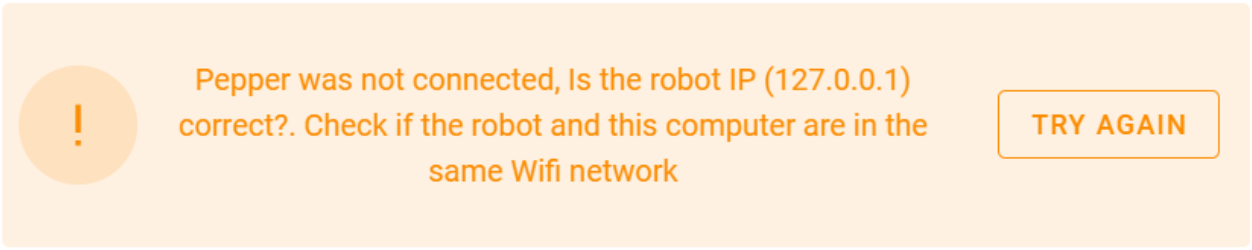


Figure 6.2: Alert displayed when the robot is not able to be connected

A set of tabs can be used to navigate between interfaces of RIZE (4). These interfaces are composed of a home page, and four programming pages representing the main types of robot behaviors that the end user can edit. Figure 6.1 shows a Pepper oriented interface. Connection with Pepper robot can be performed pressing button shown in (6). This connection button is inside a graphical element know as alert, where the connection status of robot is displayed. This element can have different colors and instructions. An example is shown in figure 6.2, where a warning message is displayed when the robot was not able to be connected.

A foot menu is used to start, pause and stop a RIZE program (7) and show the status of the interface and robot (8). For Pepper robot, an additional button must be pressed to stop all default robot behaviors (9) just after this robot is turning-on. This step is relevant, as many functions of Pepper are by default blocked. Therefore, the button shown in (9) is used to unblock Pepper functions. One of the most relevant usability issues in dataflow and block-based VPL is to deal with very large programs that require the use of many graphical elements to create complex systems. As the end-users program becomes larger and more complex, it increases the difficulty for someone (even the project owners) to read the program or search it for a particular piece of code. Even if end-users try to organize their programs to not overlap graphical elements, these are often displayed in only one workspace. Therefore, behaviors are often very difficult to read and search. Moreover, most EUD and EUP tools for robotics do not offer any searching features; if someone editing or reading a block-based or data-flow program needs to find a particular robot behavior or module, he/she must look through all graphical elements of the entire program to find it. In order to overcome this problem robot behaviors in RIZE are organized in lists with separated workspaces. Each robot behavior or item in a list is composed of an identifier (ID) and a comment, which end users can provide to explain the general functionality of each behavior. With this approach, end-users can easily search for specific robot behavior using their ID and comments. Comments are can also be used by end-users to give some hints of the functionality of each behavior to new users. An example of how robot behaviors are organized in RIZE is shown in figure 6.3. For element of the list, buttons enabling the edition of the robot's behaviors and their comments as well as enabling them to delete them are also provided.

When opening each behavior a programming environment, such as shown in figure 6.4, is opened. In this interface, end-users can design and edit the desired robot behaviors. Programming interfaces in RIZE provide mechanisms of error prevention and recovery, such as confirmation options before performing some destructive action, undo and redo buttons. The menu in the bottom is used to execute and debug the robot behaviors as well as see the code generated (in the JSON format) for each behavior.

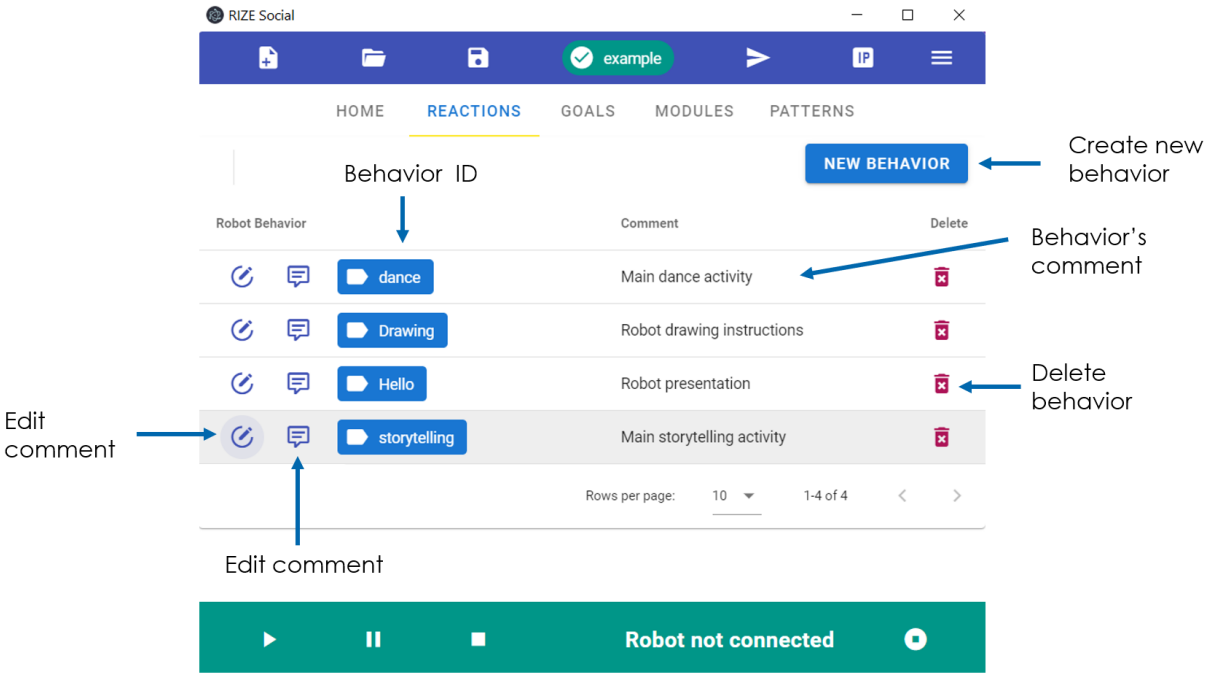


Figure 6.3: Organization of robot behaviors

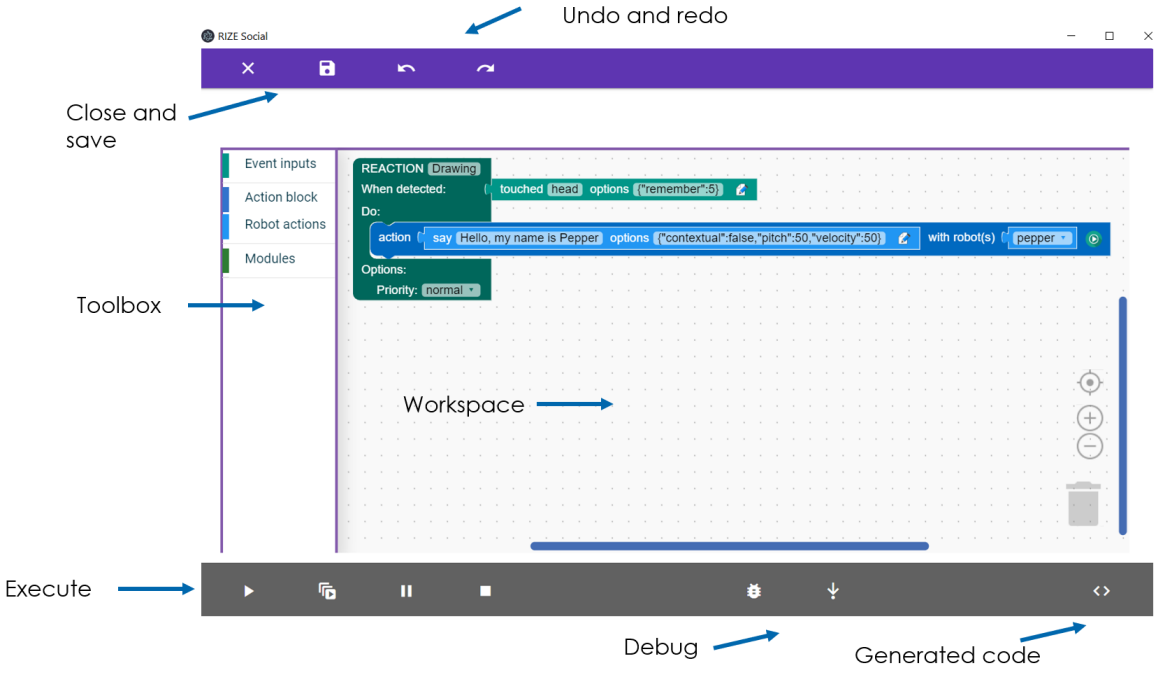


Figure 6.4: Programming environment

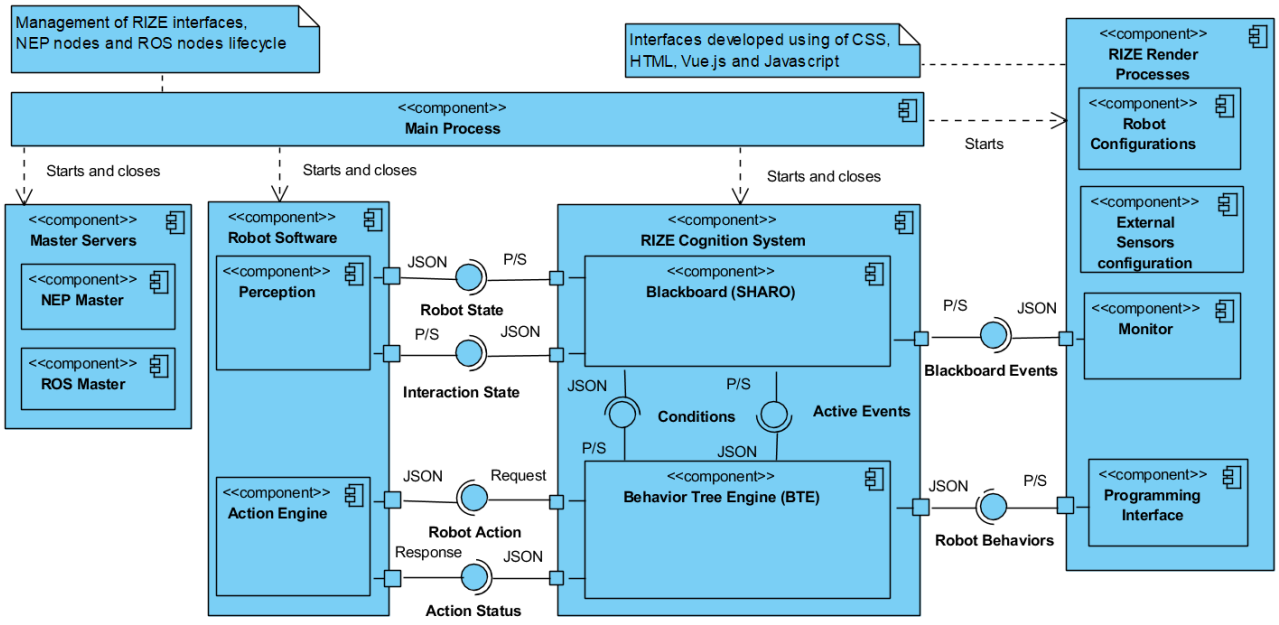


Figure 6.5: Component diagram of RIZE software architecture

6.3 Software architecture

RIZE is a cross-platform desktop application developed with modern web technologies. RIZE is based in *electron*, which combines *node.js* (an event-driven and asynchronous Javascript runtime used to create server-side applications) and *Chromium* (a lightweight open source browser) to enables the creation of cross-platform desktop application which can be installed from user-friendly desktop installers (i.e., .dmg, .exe, and .deb). This avoids the use of command line and source code compilation tasks.

The component diagram in Figure 6.5 shows how the final software architecture of RIZE works. Components in the RIZE architecture are mostly communicated using the classical *Publish-Subscribe* (P/S) pattern. Messages between nodes are serialized using the Javascript Object Notation (JSON). Applications using *electron*, as is the case of this thesis, are basically composed of a *main* process and one or more *render* processes. While the *main* process is in charge of creates and executes the *render* processes, the *render* processes create *Chromium* windows in which any web-based content can be executed. These *electron* processes can also access to system files and spawn other processes using *node.js* features. The current version of RIZE is composed of two *render* processes, the first one providing the robot programming environment and the second one enabling the logging and monitor of the robot perception and actions detected in a memory system denoted as *blackboard*. Nodes composing the robot software and external sensory devices are launched from the *render* process using the provided interfaces.

When the RIZE application starts its execution, the *main* process launches the NEP master server and ROS master server (only in ROS 1.0 compatible OS). These nodes are required to communicate the building blocks of robot architectures. Another component launched by the *main* process is the RIZE cognitive system, which is composed of a blackboard system denoted as SHAred Robot Objects (SHARO) and a Behavior Three Engine (BTE). Blackboard systems are behavioral design patterns mainly used in the video game industry to enable the collaboration of the multiple modules involved in the creation of intelligent agents. This collaboration is done by saving the output data of knowledge generators (i.e., perceptual and action modules)

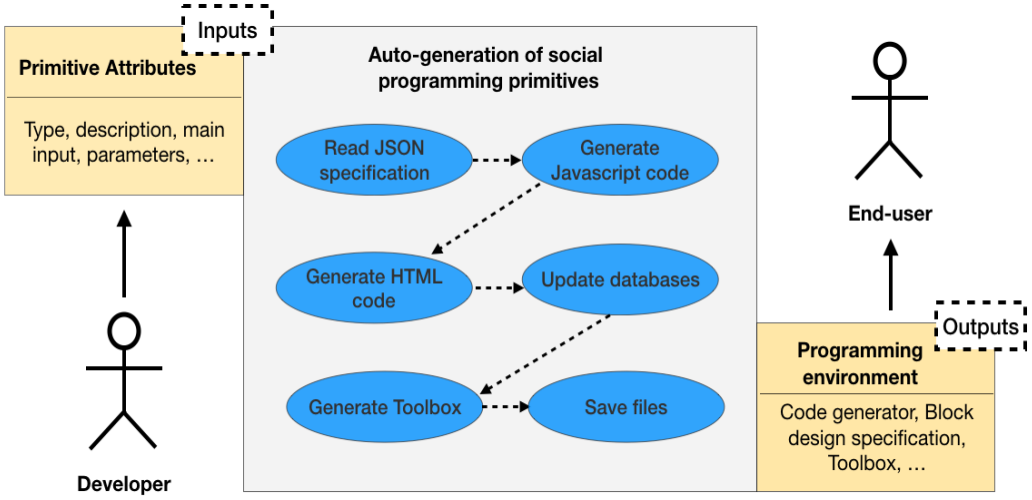


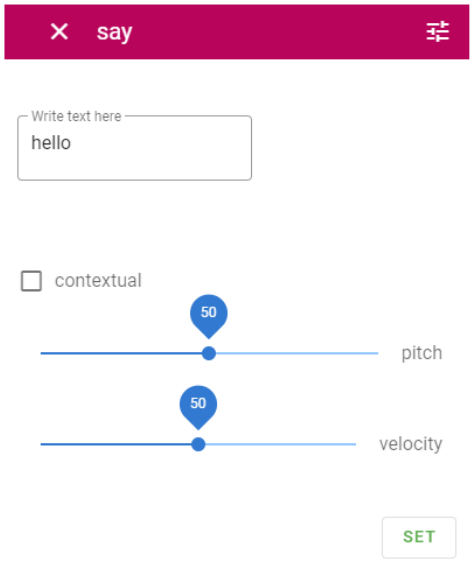
Figure 6.6: Schematic overview of the processes involved in the generation of new functional requirements for the RIZE environment

and sharing this data to other cognitive processes. Information saved and shared on blackboard systems often requires the definition of its elements using some data structure or format. A suitable way of defining this data is using *key-value* pairs. While a *key* is a string that indicates the name of the data variable, a *value* can be represented by any data type (e.g., list, dictionary, string, float, string). For this we select JSON, which is a popular and open-standard file format used in many programming languages and libraries to transmit and save data objects represented in *key-value* pairs. On the other, the BTE decodes and executes the robot behaviors, which are encoded as BTs and are defined by end-users via the RIZE programming environment. Decision-making in BTs performs on base the data available in the blackboard and the status of current nodes in execution. Finally, the module denoted as Action Engine (AE) manages the execution of the robot actions that are specified to be executed by the BTE. Finally configuration of robot and external devices are also saved as JSON files.

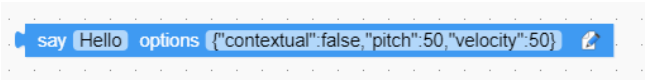
6.4 Automatic Generation of Behavioral Blocks and Code

In order to reduce the difficulty of supporting new functional requirements in EUD environments for Robotics, it was created a Python tool allowing for easy maintenance of visual elements in the RIZE interface. The main idea is to define a set of usable and flexible social abstraction primitives for robot programming. The specification of these primitives is made in JSON files. These files are read by a script that generates the JavaScript and HTML code that defines the design and the form interfaces and of each visual blocks, this script also defines the Python and JavaScript code that must be generated when the user drag and drop each block in the visual programming environment, and updates the needed XML and JavaScript files needed to compile the Google Blockly library. As shown in figure 6.6, developers which require to add new functions to the RIZE programming environment must define the main attributes of these functionalities using the JSON format. These specifications are then read by a Python script that auto-generates the JavaScript and HTML code defining the design and functionalities of each behavioral block in the Google Blockly environment. The Python tool also update the databases and toolbox of RIZE programming environment.

As an example, let's suppose that a developer needs to add the speech capability shown in figure 6.7 to the RIZE interface. After developing the nodes needed to support this capability in



(a) Form interface



(b) Programming element

Figure 6.7: Example of auto-generated graphical elements

a component-based framework such as ROS, the developer can specify the parameters of social skill such as is shown in listing **??**. In the JSON notation, the developer defines an intuitive title for the social skills, which will be visualized in a block element. In this case, the speech capability is denoted as an imperative action *say*. Then the developer needs to specify the type of social skill as input (perception) or output (action). Developers can also add a description of the block's functionality to help users. This description will be dynamically shown when the user selects the respective block. The *input* file is used to specify the main parameter of the block, in this case, a string that indicates the text to be said for the robot. The developer can also specify a set of optional inputs which are parsed to generate the respective HTML and JavaScript code that creates a form-based interface for each primitive. An example of the auto-generated form-based interface for the say primitive is shown in figure 6.7.a. Form-based interfaces can be opened by the end-user pressing the button in the primitive's block. Form elements that can be automatically generated are sliders for numerical inputs, a textbox for string inputs, checkboxes for Boolean inputs, drop-down lists for the selection of database elements (e.g. a list of emotional expressions, sounds, or animations that are available in the robot), among others.

6.5 Graphical elements

This section describes the main graphical elements that RIZE provides for their use by end-users.

6.5.1 Definition of robot actions

Robot action primitives can be executed and debugged using the action block presented in figure 6.8. This block presents a input field for primitives and other for robots. In the primitives field

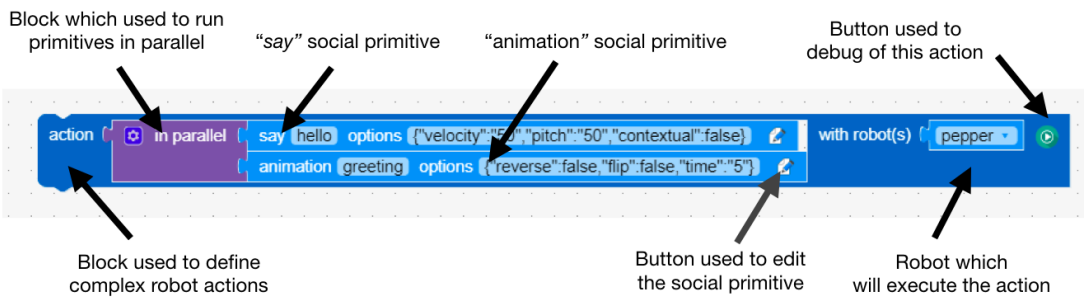


Figure 6.8: Example of an action in RIZE

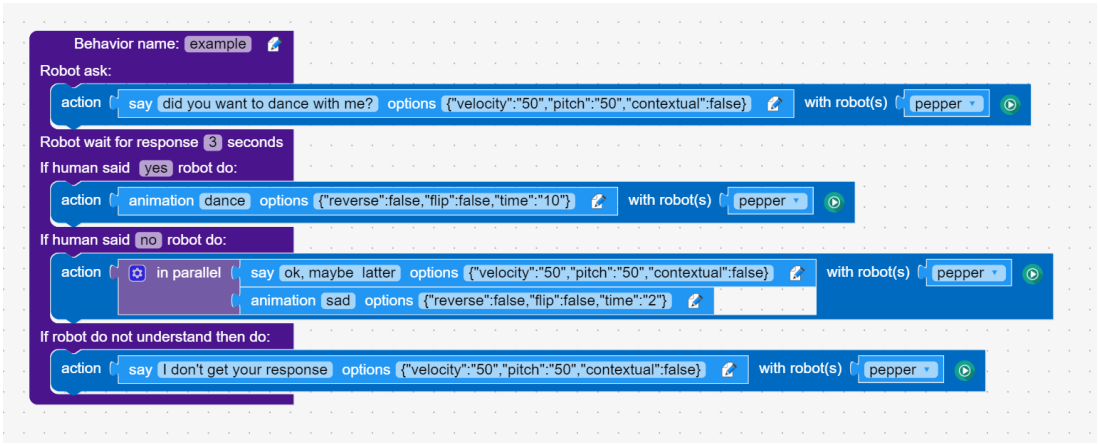


Figure 6.9: Example of a pattern block

the user can concatenate a set of primitives in a list, this send the message to execute the primitives in parallel. The robot field can also takes a list of robot to perform the defined action in multiple robots at the same time. When the user press the button in the action block, this launch a signal to execute the defined action. Therefore, the end-user can execute all the program or only an specific part of the program developed for testing proposes.

6.5.2 Interaction patterns

In order to avoid hard mental operations, the definition of classical interaction patterns [207], [256] is done with the help of some pre-defined blocks denoted as behaviors. An example is shown in figure 6.9, which shows the block that is used to define a question-wait-answer pattern [207]. In this block a set of actions can be selected to ask something to the human, then some time defined by the user is selected to enable the response of the human, finally, the response can be processed in base the detected human speech.

6.5.3 Modules

In RIZE a list of actions or behaviors can be encapsulated in modules. These modules can execute all their elements in sequence or only execute one of their elements randomly. These decision-making operations correspond to the Selector and Random composite nodes in a BT. An example of a module is shown in figure 6.10. This module is composed of two actions that are set to be executed in descendent order.

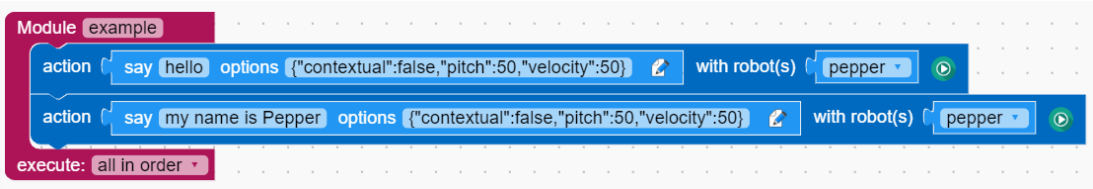


Figure 6.10: Sequence module blocks example

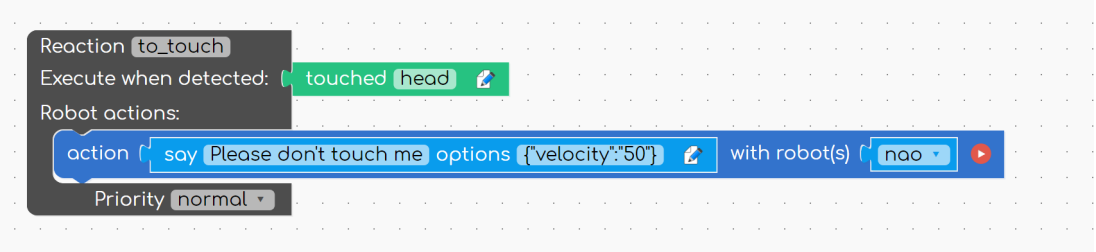


Figure 6.11: Reaction block example

6.5.4 Definition of robot reactions

These modules are small sub-BT which can be activated after the presence of a particular social stimulus. An example modeled using the RIZE interface is shown in figure 6.11. In this example, the action of text to speech is defined to be executed every time the robot is touched in the head (the trigger condition). The proposed approach not only enables the creation of Trigger-Action Programming like [171] (if-then) behaviors but also enables to define the priority of these behaviors. This priority value is also used to preempt other modules that are in execution.

6.5.5 Definition of robot goals

Authoring plans are defined as sequences of actions, or other complex behaviors. A goal is activated or canceled if some sets of conditions are true. Unlike reactions, goals can continue its execution after their preemption by some reaction with higher priority. In the proposed approach only one goal or one reaction can be executed at the same time. A simple example of goal behavior is shown in figure 6.12. In this example, the goal is activated until 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 when the behavior is canceled the robot will say "*Ok, I will stop*". Moreover, if this behavior is preempted to attend a high priority reaction, the flow of the program can return to the goal and continue with its execution. Furthermore, before executing the missing actions of the goal, the robot can execute some action to notify humans that he will continue with the original goal. In the example shown in figure goal_block, this return behavior is modeled as a say action with the text "*As I was saying*". This relatively complex behavior is surprisingly very difficult to model in data-flow interfaces such as Choregraphe [190].

The approach used to execute reactions and goals is similar to the BT with memory method [148]. However, the proposed approach uses the FSM defined in the high-level layer of the proposed BTE to enable reactivity. For this, a register of the conditions that activate or cancel the defined reactions or goals as well as their priorities is first done. A program in RIZE maintains its execution in the *idle* state until a set of social primitives that activate some goal

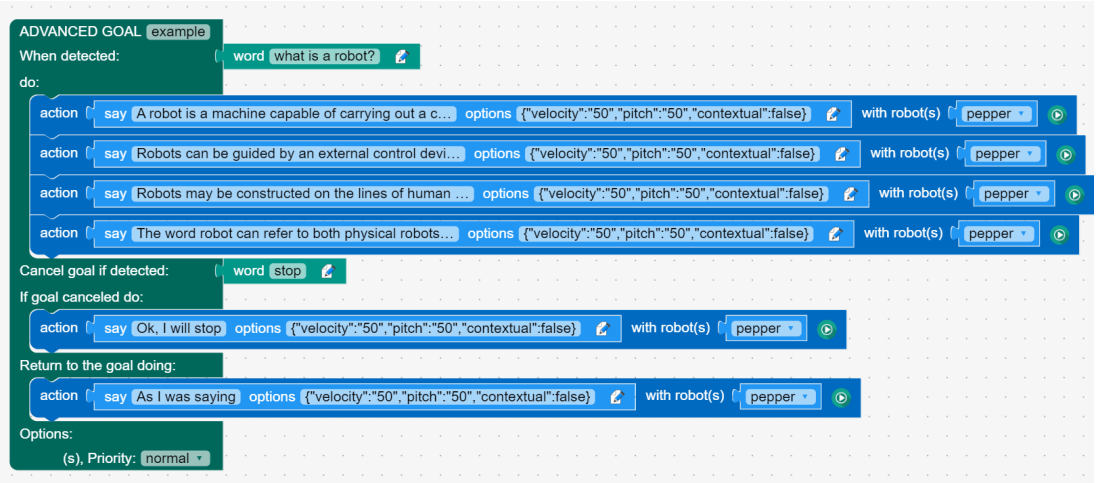


Figure 6.12: Goal block example

or reaction is detected in the blackboard (see figure 7). 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. In the proposed approach, conditions that activate reactions or goals with the same and lower priorities of the behavior in execution are not checked. 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 completed (RA is true when GA is true), this goal can continue its execution after the reaction that caused the preemption finishes its execution (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 to *idle*.

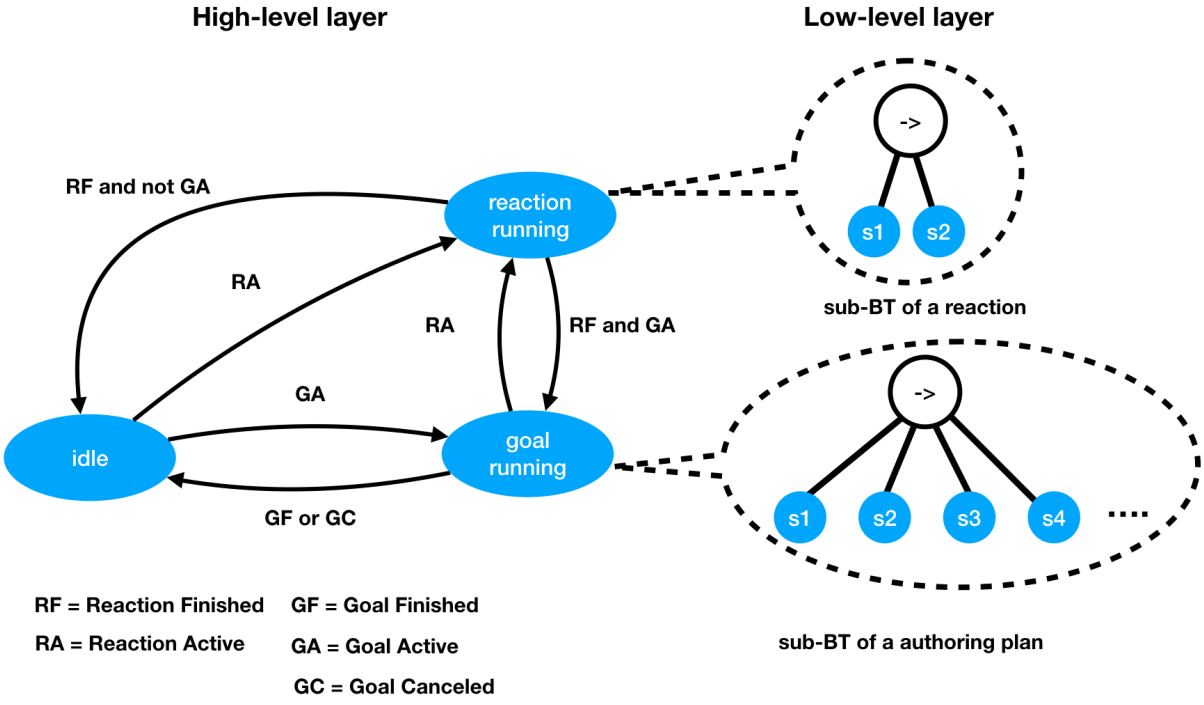


Figure 6.13: Proposed hybrid FMS and BT engine approach

6.6 Summary

This chapter describes guidelines, features and software architecture of RIZE, a modular VPE for "in the wild" human-robot interaction research. Proposed architecture enables the creation of intelligent and autonomous social robots, which can be designed by both expert programmers and domain expertise users using modular and easy to use software tools. The proposed method enables the modeling and execution of reactive behaviors robot behaviors. Moreover, the proposed hybrid approach avoids some of the disadvantages when only using pure FSM or BT decision-making engines. Finally, developers can easily add more robotic functions using proposed tools for generating code and visual elements.

Chapter 7

Long-term Human-Robot Interaction in Domestic Scenarios

This chapter presents an example of an interdisciplinary HRI research project supported by the two main software tools proposed in this thesis. First, the motivations and objectives of this project are briefly defined in section 7.1. Then, section 7.2 describes the proposed software architecture, which is based in NEP. Section 7.3 describes the experimental sessions summarize some relevant HRI data obtained from implemented robot perceptual capabilities. Finally, section 7.4 concludes presenting: (i) advantage of the proposed software, (ii) difficulties when performing experiments in domestic environments and (iii) limitations of the proposed implementation.

7.1 Objectives and motivations

Unlike experiments in most service and public spaces (e.g. stores, hotels, hospitals or theaters) where people and robots share the same space in short-term interactive sessions (e.g. 5 minutes, 1 hour), experiments in home environments require that robots can be able to play complex social roles in long-term, interrupted and unsupervised sessions (e.g. one day, one week or one month). Robots in domestic environments must coexist in the intimacy of the experimental subject's homes. Therefore, the option of sending researchers to the experimental settings to perform data collection and direct monitoring of the robot and experiments disrupt the dynamics and naturalness of the presented interactions [257]. Moreover, the use of video or audio streaming to enable Wizard of OZ approaches for controlling the robot behaviors, such as presented in [258, 259], can be tedious or even unethical tasks. Therefore, novel methods for data collection and generation of intelligent and autonomous robot interactions must be explored.

Recently, some efforts have been done to bring social intelligent robots in homes. Some of them have focused on the development or integration of high-performance and complex robot functions, such as, navigation, object recognition, manipulation, among others. Relevant examples are [260, 261]. However, most of these experiments have been designed by only expert programmers and mainly oriented to be evaluated in simulated environments (i.e. virtual, exhibition, competitive or laboratory settings) rather than in real scenarios. While experiments in simulated environments are useful to demonstrate and validate the possible capabilities of social robots, they often provide poor information about those HRI aspects able to enhance the acceptance of social robots. Some of the most relevant are usability and user experience [262, 257]. These types of experiments cannot be used to fully understand the challenges that involve really coexisting with robots in homes [257].

In order to provide valuable insights about those non-functional and design aspects able to provide enjoyable and valuable user experiences, social HRI experiments in domestic environ-

ments need: (i) to be designed by cross-disciplinary teams rather than only expert programmers and (ii) be performed in real home environments rather than in simulated settings. Even with the current advances in sensory, perception, navigation and control algorithms, only very few studies of long-term social interaction in real homes have been reported [239, 263, 264]. In fact, this type of studies requires time-consuming developing, experimental setup (e.g. recruiting participant, perform interviews and transportation of equipment), design and testing processes before implementing the real experiment. From the few multidisciplinary teams that have performed "in the wild" experiments in homes, most researchers agree that "research robots are not robust enough to be used in prolonged experimental sessions without expert supervision" [265, 239]. Therefore, most of these studies have used limited functionalities and structures, robot toys or commercial robot vacuum cleaners. Examples are described in [266, 267]. Very few works recently presented in [268, 269, 264] have used more complex robots. However, works reporting the use of component-based, open, robot independent, user-friendly software tools that enable cooperation between domain specific experts and robot experts for performing HRI research in domestic research are still rare.

The objective of this project is to provide more valuables insight towards the development of companions robots. This project was proposed by experts in design and user experience. The work done in this project was mainly focused on supporting these domain specif experts and enable them to develop their desired applications with social robots. For this, an initial version of RIZE was used.

7.2 Software architecture

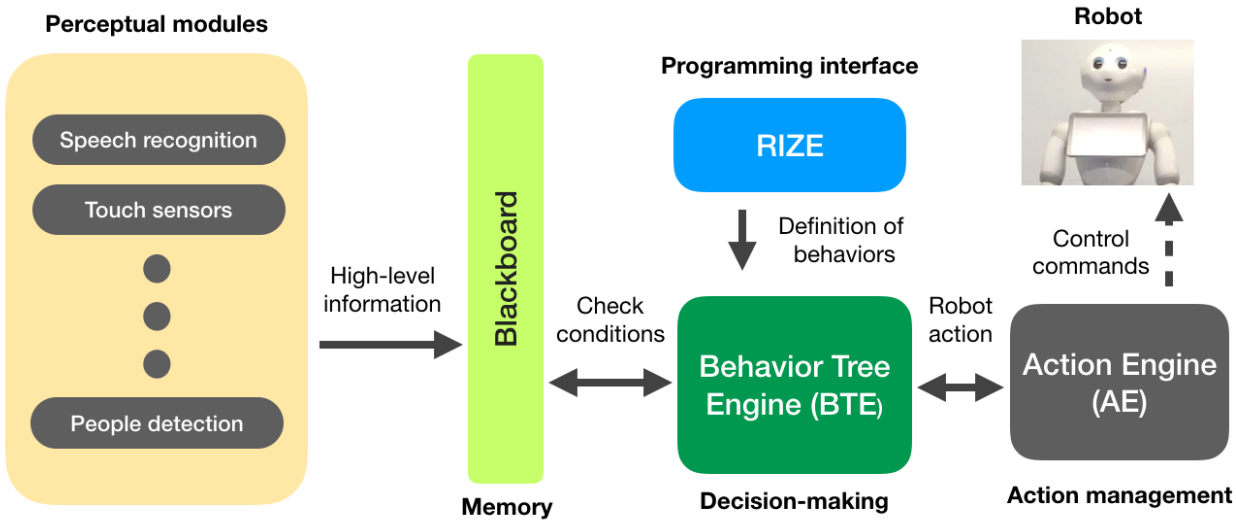


Figure 7.1: Proposed software architecture used in experiments performed in domestic environments

The proposed software architecture is basically composed of perceptual modules, a Blackboard system, a Behavior Three Engine (BTE), an Action Engine (AE) and the RIZE programming interface. Figure 7.1 shows how these basic modules interact together to create robot applications. We use the *Publish/Subscribe* pattern to broadcast perceptual information using an event-based approach. In the proposed approach, perceptual nodes obtain the low abstraction data as inputs and generate output values as social primitives or skills [156]. These outputs are encoded using the JSON format and sent to the blackboard systems, which temporally save the data for posterior processing. Included perceptual features, such as speech

recognition, detect people and obtain data form touch sensors were implemented using the official development kit of Pepper robot. As introduced in chapter 6, a blackboard is a behavioral design pattern often used to create intelligent virtual agents in video games. The developed blackboard has two main tasks: (i) collect output published by perceptual nodes and share this relevant information with the decision-making engine (ii) save and send these perceptual outputs by email for remote supervising the state of the robot. End-users modeled the robot behaviors using an early version of RIZE. The design of this version is shown in figure 7.2. In this initial version of RIZE behaviors were organized in two Workspaces separating main available types of behaviors: *goals* and *reactions*. These types of behavior were defined in chapter 6. This version only included some basic options enabling users to create, load, save and download their program. Rather than use electron, Vue.js and Node.js (web technologies used in the final version of RIZE and described chapter 3), this interface was developed using: (i) Flask, a server-side library written in Python that enables to load HTML, CSS and Javascript code in some browser; and (ii) Boostrap, a Javascript and HTML framework containing a set of templates enabling the creation of button, forms, labels, checkboxes and others elements for the design of the interface of a web application. Finally, an early version of the proposed decision-making and action execution nodes were used. Unlike the final version described in chapter 6, which was developed in JavaScript, this pilot version was develop in Python. Communication between the the decision-making and action-making nodes was performed using the *Survey* pattern [270]. This pattern is similar to Publish/Subscribe in that a process, denoted as *Surveyor*, can broadcast messages (not aimed at a particular peer) to a group of processes. However, each process in that group, denoted as *Responder* nodes, can call back to *Surveyor* process. Unlike *Client/Server*, this pattern is non-blocking, which avoids the occurrence of deadlock issues. The *Survey* pattern allows easy development of applications where a query of the state of a large number of components is needed. This is, in fact, the same nature needed for executing BTs, where the *Surveyor* (i.e., a BT-based program) performs queries to some *Responder* (i.e, a process in charge to execute the robot behaviors) each time a tick signal is generated. This *Responder* must return *success*, *failure*, or *running* depending of the execution status in a non-blocking way. This patter is only available in the nanomsg message-library. However, this library can be very difficult to install in OSX and Linux and in some computers with Windows. Therefore, the approach used to communicate decision-making and action execution nodes were replaced by that presented in chapter 6.

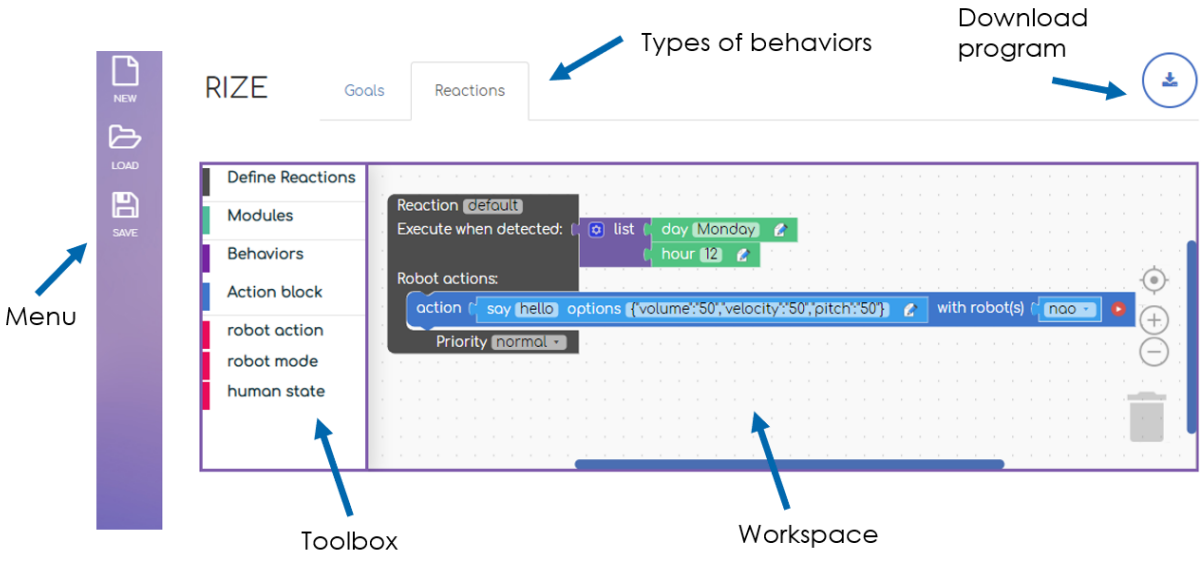


Figure 7.2: Initial version of RIZE

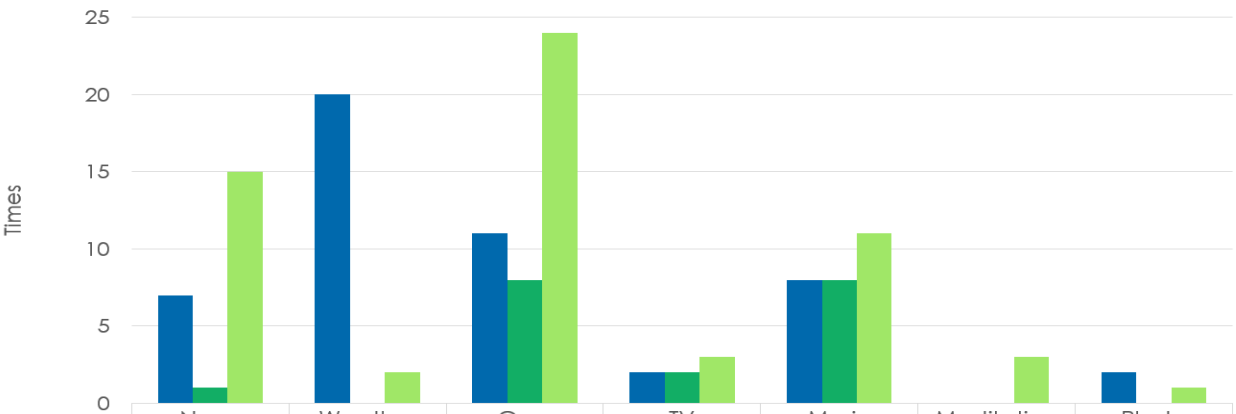
Figure 7.3: Example of "in the wild" scenarios used for experimental sessions

7.3 Validation "in the wild"

In order to meet the research objectives of interaction designers as well as to test the suitability of the proposed architecture and, some experimental sessions in real Japanese domestic environments were performed. Functionalities integrated for this application were text to speech, gesture expression, touch detection, speech recognition, reproduce video in the tablet, take pictures, show weather, show news, show TV schedule, change of led colors, distance detection, overheating detection, people detection, sound reproduction, among others. Navigation skills were not allowed by the experimental subjects due to the dimension of the robot used and the small space in most of the real homes tested. Images of the real home experimental settings are shown in figure 7.3. Robot's intelligent behaviors were defined by an interaction design expert using RIZE rather than an expert programmer. Examples of activities developed by the end-user are: perform together with the exercises, give some information about some specific topics, take a break and meditate with the robot, shown new and weather, take pictures among others. Figure 7.4 shows the total times experimental subjects utilized each of the robot's functions.

7.4 Discussion

Even when this project was performed using a very early version of NEP and RIZE, experimental results proved the technological suitability of the proposed software. This chapter only presented the operational features of the proposed architecture, which enabled a humanoid robot to perform in real domestic scenarios. These experiments are part of a long-run project involving experimental sessions that aim to be performed not only in Japanese but also in French homes for cross-cultural comparisons and analysis. This project is mainly conducted



	News	Weather	Gym	TV	Music	Meditation	Photo
Session 1	7	20	11	2	8	0	2
Session 2	1	0	8	2	8	0	0
Session 3	15	2	24	3	11	3	1

Figure 7.4: Total usage of robot most used functions over the course of the three sessions

by social researchers. Therefore, the description of their objectives and findings is out of the scope of this thesis. However, this chapter proved how RIZE can be used for users enabling end-users to conduct their own experiments and research activities in HRI.

Chapter 8

EUD of Children-Robot Interaction Applications using RIZE

Robots are no longer confined to work in laboratories, they are also becoming an essential tool for social Human-Robot Interaction (HRI) in real-world and every-day environments [21]. In this context, one current trend is Children-Robot Interaction (CRI). This topic is especially popular in the areas of Robot-Assisted Therapy (RAT) and to deal with other health issues. Examples are shown in [271, 272, 273]. Most CRI studies reported in the literature perform in highly structured, controlled and static scenarios inside laboratories. This approach makes data collection more manageable and avoids many of the technical issues often presented when robots perform in open, uncertain and highly dynamic environments. However, the HRI community has recently expressed the necessity to move towards natural, open, everyday environments: an approach referred to as "HRI in the wild" [49, 50]. The importance of "in the wild" research is in the acquisition of more valuable quantitative and qualitative information, which can be used to improve the design of robots and their applications, therefore increasing their economic and social value [51].

This chapter shows how NEP and RIZE can be used to develop more robust, easy-to-use and re-usable robot software and applications for CRI that are able to perform "in the wild". Experimental validations of the proposed tools are performed in open, noisy and dynamic scenarios (kindergarten and public events) using Pepper and NAO humanoid robots. For this, four different tasks (storytelling, dance, game, and autonomous interaction) were designed and programmed by a real end-users.

8.1 Software architecture

The used software architecture is an updated version of that presented in chapter 7. This new version was modified to provide some remote-controlled functionalities. This is done to deal with some of the required interactive tasks, which are hard to develop with the current state-of-art perceptual algorithms and due to the highly dynamic and noisy nature of the environments of these CRI projects addressed. As shown in figure 8.1, a node that acquires and processes data from a game controller is used as an additional perceptual output. This data is obtained by the blackboard node for their posterior processing by the decision-making engine. Modeled behaviors executed in the decision-making engine are defined using RIZE. Unlike the approach used in application performed in home environments (chapter 7) and the final version of the RIZE software architecture, the decision-making and action-making (robot controller) nodes are connected using the *Client-Server* pattern. This approach enables the easy install of RIZE but sometimes produced deadlocks, which requires to restart the control architecture. However, these issues were only presented few times in pilot tests and not in the final experimental

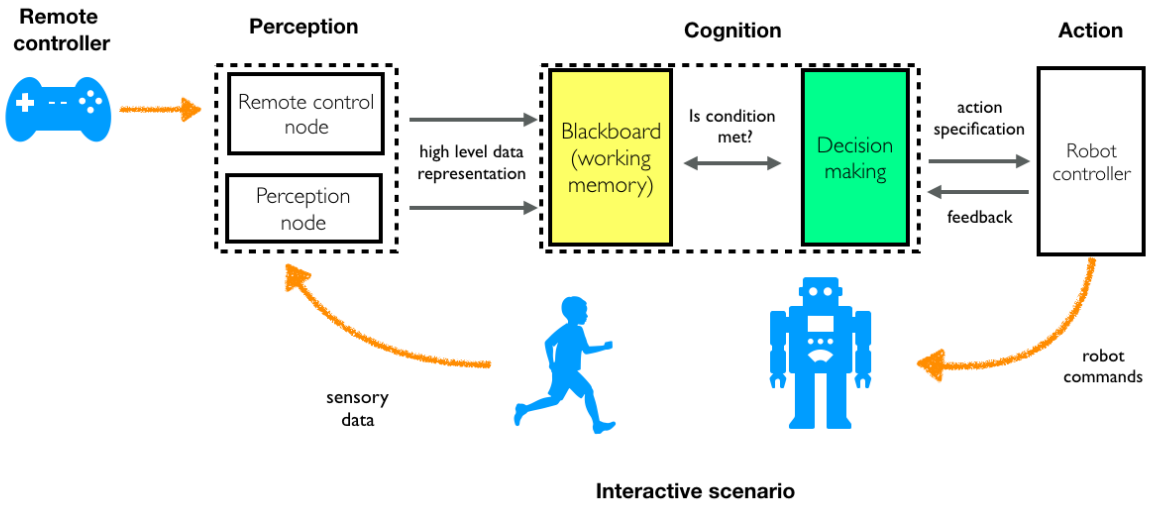


Figure 8.1: Software architecture diagram used for supporting Children-Robot Interaction (CRI) applications

evaluations. In order to improve the robustness of the system the final software architecture of RIZE only used the *Publish/Subscribe* pattern as presented in chapter 6.

8.2 Activities designed by end-users

This section briefly describes the activities designed and developed by end-users for supporting their research and professional goals. Figure 8.2 shows an example of how end-users used RIZE for developing their own applications with social robots.

8.2.1 Storybook reading

For this activity, the robot had to be able to execute a sequence of audio and speech actions accompanied by images displayed on its tablet as part of two distinct storybooks. These storybooks were peppered with questions designed to engage the children, modeled after an experiment held previously at another preschool. Rather than only enable the execution of a script, reactive behaviors can be activated autonomously based on stimuli detected by the robot's sensors and the decision-making performed by the cognitive node. Examples of reactive behaviors are actions like say *"please step back to hear the story"* when children draw too close to the robot, or *"please don't touch me now; I'm telling a story"* when someone touches the robot. After launching one reactive behavior, the robot can then continue from the last action in the interrupted sequence. These types of behaviors, which are cumbersome to model in dataflow interfaces, can be easily defined in a modular way using the RIZE interface. Figure 3.a shows the implementation of this activity. For this activity a set of images from kindergarten story books were displayed in the tablet attached to the robot.

8.2.2 Game

The Japanese version of the game "Red Light, Green Light" was programmed to be performed by the robot. In this game, the children initially form a line side-by-side approximately 10 meters away and facing the robot. The objective of the players is to touch the robot. However,

Figure 8.2: Example of real end users designing and programming a "*in the wild*" Human–Robot Interaction (HRI) scenario using current prototype of the RIZE robot End-User Development (EUD) interface for social robots

players can only move at certain points in the game. In the Japanese version of this game, the player at the front of the room typically turns around and utters the phrase, "Daruma-san ga koronda", only turning to face the other players once the phrase has been completed. Any player who is seen moving by this player is out. After a beta-test with the robot moving its head to look up or down proved to be too slow and therefore confusing, we adapted it so that instead of looking away, the color of the robot's LEDs changes to indicate to the player when it is okay to move. When a player touches the robot, the game is over. For this game, the robot had to explain the rules, change the tempo of the phrase uttered, and indicate when a player has moved at the wrong time and is therefore out of the game. This activity presents a combination of autonomous and remote-controlled behaviors due to the non-structured environment. Figure 8.3.b shows the implementation of this activity.

8.2.3 Dance

Rather than execute improvised and creative movements by the robot (such as proposed in [274, 275]), this activity was designed to follow a sequence of well-defined movements so that the robot could take part in a dance with which the children were already familiar. This scenario differs from typical CRI settings, where the children must repeat a set of movements taught to them by the robot in a sequence that can be controlled by the Wizard of Oz method, such as [276]. Because the robot is participating in a dance that the children already know, the children would notice if the robot were to fall out of sync with them in the dance. For this, we adopted a semi-autonomous approach, where synchronization is remotely assisted by a human. The movements selected for the dance were defined by kinetic teaching as a set of primitives which are concatenated as a sequence of elements in a BT, and which can be easily reused for future experiments and other dances. These motion primitives can also be easily configured to optimize the execution time, which is the most relevant parameter for this application. Figure 8.3.c shows the implementation of this activity.

8.2.4 Other activities

End-users developed a set of relatively simple but relevant activities with Pepper and NAP robot that enable these robots to conduct research experiments in CHI. Some variants of storytelling activities were implemented. In these variants Pepper did not use the tablet (figure 8.4.a) or were implemented in NAO 8.4.b. Another end-user activities required the use of Wizard of OZ

(a) Storytelling (b) Game (c) Dance

Figure 8.3: Example of activities designed by end-users using RIZE for a kindergarten event

(a) Storytelling using NAO robot (b) Storytelling without using the tablet

Figure 8.4: Example of variant of dance and storytelling activities used in other events

paradigm for remote control the robot and response questions done by children (figure 8.5.a), shows specific poses to observe the children reactions (figure 8.5.b), and conduct interactive activities (figure 8.5.c and 8.5.d).

8.3 Experimental insights

While most of data obtained from experiments were collected by end-users for their research goals in social science. One of these experiments was conducted by PhD students of GVlab. In this application a questionnaire was handed out to parents, requesting their comments and their children's opinions of the experience. Out of the 30+ children present, 12 questionnaires were filled out and returned, representing 15 children of ages ranging from 1 to 6 (mean 4, median 4, mode 3). The responses were largely positive, with many parents expressing that their children were delighted that the robot was able to dance a familiar dance with them. Negative comments mostly revolved around children being disillusioned about the autonomy of the robot due to the interaction not being natural enough (i.e., they expected to be able to run around freely with the robot and converse with it naturally), or due to seeing the robot being controlled with the laptop.

The questionnaire contained the question, "How does your child perceive the robot?" with answer options being "Another child", "A toy", "A pet" or "Other". 7 respondents said "A toy", with one respondent adding that this particular child felt that someday he could be friends with Pepper; of the remaining respondents, 1 replied "Another child (a friend)", 1 replied "Other (a presenter)", 1 replied "Other (a robot)", and 1 replied "Other (scary)".

(a) Children asking questions to Pepper

(b) Error activity, reaction of children

(c) Pepper conducting drawing activity

(d) Children showing their drawings to Pepper

Figure 8.5: More popular messaging patterns

Technical issues

Tests performed in open and semi-open air spaces (such as was the case for this event) with a large number of subjects increases the technical challenges of experimental settings. One unexpected issue was the voice volume of the Pepper robot, which even at maximum volume was inaudible in the experiment environment. Due to this, an external microphone had to be held up to one of the robot speakers. As a result, many children seemed to become fixated on the thought that "Pepper speaks with its ears" (as speakers are located on each side of its head).

Strong rays of sunlight made it impossible to see the light changing colors in the robot's eyes. Due to this, no child moved in the first attempt at the game, and this parameter had to be hastily changed to the LEDs on the robot's shoulders: they would turn on when the children could move, and off when the children could not. Nevertheless, whether because it was still difficult to see or because they were not used to the participant at the front of the room not looking away, the children did not seem to use the lights as an indicator of when to move. They moved only while the robot was uttering words, and when the robot paused in the middle of a phrase, they paused their movements as well. Only after several interested children had played the game multiple times did they start daring to move during these pauses in the phrase.

The effect of novelty

During the first storybook reading, the children seemed utterly uninterested in anything Pepper was saying. There was no reaction to any of the robot's behaviors meant to initiate interaction; the children instead seemed more intent on touching and feeling the robot. Due to a combination of the volume problem mentioned above and poor timing of the microphone being pulled away from the robot's speakers and a measure added to the program to prevent the robot from repeating itself and therefore becoming annoying, the robot's request for the children to step away and stop touching it went unheard. After a minute so had gone by and the robot's request for space was audible, not one child reacted. Even adults trying to convince the children to back up a few steps only gained a begrudging ten centimeters or so of space. Therefore, the first storybook functioned more like a session for the children to touch the robot and become familiar with it.

After having had some time to touch the robot, and having played the game with the robot, the children were a little more receptive to the robot during the second storybook. They still did not react to the robot's interactive phrases designed to engage them; however, this time, when the robot would state that they were too close and could they please back up, the children would obey instantly without the need for human intervention.

8.4 Discussion

This chapter described some of the most relevant CRI applications developed using RIZE. Unlike similar applications and approaches found in literature, developed applications were performed in by real end-users and executed in open, public, unstructured, dynamic and noise environments. Examples shown in this chapter can be used to ask the research question RQ8. This examples proved the suitability of RIZE for enabling the creation and execution of CRI applications performed "in the wild". Feedback from end-users and presented technical issues in pilot tests were used to improve the presented software architecture of RIZE. Finally, data obtained from one of the experiments prove the suitability of proposed activities which can be considered as an initial step towards the creation of more valuable experiences with social robots.

Chapter 9

Building Emotional Intelligent Robots with NEP and MATLAB

In order to convey a sense of believability in social contexts, new interactive robots may be able to express dynamic expressive states and to adapt to different situations. This chapter presents an example of an advanced control architecture for generating expressive motion in robots. The proposed architecture was developed using NEP and updated for different demonstrations performed in laboratory settings by several students and researchers working in the Gentiane Venture lab (GVlab). Therefore, some of the contributions reached by researchers involving this project are not detailed. Instead, this chapter only describes those contributions done as part of this doctoral work. In this context, the main focus of this chapter is to prove the technological suitability of NEP for MATLAB/OCTAVE for supporting advanced research projects focused on developing novel control architectures for robots. As an additional contribution, this chapter presents a novel and adaptive scheme for modeling emotions in robots based on Fuzzy Logic, which uses both information from humans and the environment to change the internal and expressive robot states.

9.1 Emotional modelling using dimensional values

Emotions and mood are important elements in affective computing. While emotions are generally expressed using instantaneous behaviors such as, facial expressions or gestures, moods are longer-lived and affect the performance of the human behaviors [277]. A well known model used in affective computing for encoding both emotions and moods is the PAD (Pleasure-Arousal-Dominance) dimensional model [278], [279]. In this 3D model, general mood types are divided in eight octants and emotional states are represented as moving vectors with the current pleasure, arousal and dominance values. The *pleasure* component of the PAD model represents an affective balance and varies from positive to negative. The *arousal* component indicates the degree of physical activity and varies from exited to calm. Finally, the *dominance* component represents the degree of control or influence over the environment and varies from weak to strong [280]. Figure 9.1 shows how some emotional states are mapped in the PAD model.

9.2 General software Architecture

The objective of this project is the creation of a robotics system that can be able to adapt their expressive states (postures/motions) to the presented HRI scenarios and the current environment. This section describes a general system architecture, which use NEP to connect robot components performing perception, cognition and robot control. A relevant requirement was the connection of sensory, cognition and perceptual component with the action making

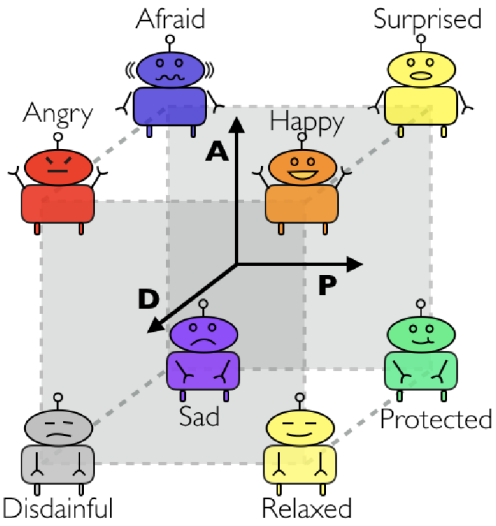


Figure 9.1: PAD emotional model, from [4]

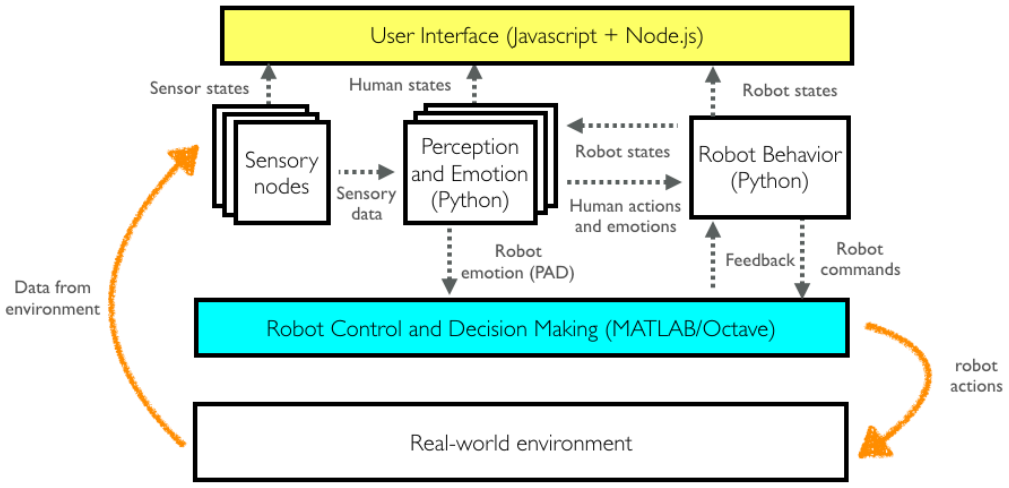


Figure 9.2: General cognitive and adaptive architecture for emotional robots

module, which is written in MATLAB. This MATLAB module use optimal predictive, and robust motion-controllers to generate expressive motions in arm robots (presented in chapter 3). The interaction between these modules is summarized in figure 9.2. The main parts of the architecture includes: i) A set of data acquisition modules obtaining sensory information from the real-world; ii) Perceptual modules, which subscribe and transform the low-level sensory data into high-level representations (e.g, gestures, human actions, behaviors, and emotions); iii) A module that manages decision-making using high-level knowledge; iv) The robot controller (written in MATLAB), which continuously receives information from sensory, perceptual and decision-making modules thought different NEP topics; and v) a web-based user interface displaying sensory, perception, and cognitive states. This user interface is based in the same software technologies used to create RIZE (explained in chapter 3). We use continuous and dimensional values of emotions to adapt parameters of robot controllers (enabling more emotional reactions and expressions).

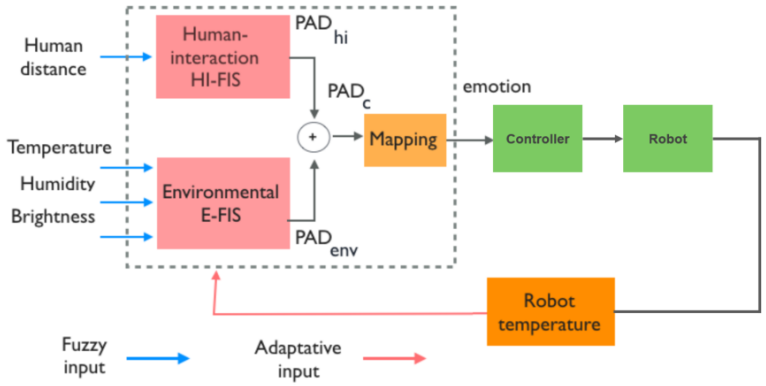


Figure 9.3: Control Architecture for expression of emotional states in robot arm

9.3 Emotional modelling using Fuzzy Logic

Affective states are always defined in natural conversation with linguistic variables. In fact, one often says things like: *"you are very happy"* or *"he seems a little bit sad"*. Moreover, these linguistic variables have unclear boundaries between them and can vary depending the human personality and the environment or the context. A powerful approach that can be used to model this uncertainty is the fuzzy logic [281]. The advantage of this technique is that it allows to define the solution of a problem in terms of unclear linguistic variables. This makes that human experience can be easily modeled. Therefore, an emotional modelling approach based in Fuzzy Logic was presented as an additional contribution in this project. The proposed Adaptive Fuzzy Inference System (FIS) is used to represent the perception and making decision in cognitive controllers. Input values of this FIS is based on the dimensional modelling of emotions using the PAD dimensional model (section 9.1). The PAD is created with the human, environment interaction and robot internal states. These inputs and adaptive parameters affect directly the standard deviation of the Fuzzy membership functions, moving the final PAD for generating the robot expressive motions.

Figure 9.3 shows the scheme of the Adaptive Fuzzy Emotional Model (AFEM) proposed. This approach is composed of two Fuzzy Inference System (FIS). These blocks model the influence that the environmental conditions and the human activities have on the affective states of the robot. These blocks are Environmental Fuzzy Inference System (E-FIS) and Human-Interaction Fuzzy Inference System (HI-FIS), respectively. In the E-FIS, the environment temperature, humidity and brightness were used as inputs. Linguistic variables used as inputs are cold, comfort or hot for the temperature input; dry, comfort and wet for humidity; and low or high for brightness. The ranges of these linguistic variables are defined based on [**?**] and [**?**]. The outputs of the E-FIS are values of the PAD model (Pleasure, Dominance and Arousal), which can take the values of low, neutral or high. The range of each PAD axis is from -10 to 10. Gaussian distributions are used for the modeling of the membership functions. The number of rules defined in the E-FIS are 18. This E-FIS system uses an adaptive function, which is modified according to the robot temperature. The output of this adaptation changes the membership functions in E-FIS generating new PAD values that produce a suitable robot motion. The main idea of this approach is to replicate in the robot the influence that the body temperature affects the perception of the thermal comfort of humans [282], which is a physiological value that can be linked to affective states.

For the HI-FIS, the input is the distance between the human and the robot. The definition of the linguistic values were defined in studies of human proxemics [283]. This variable can take

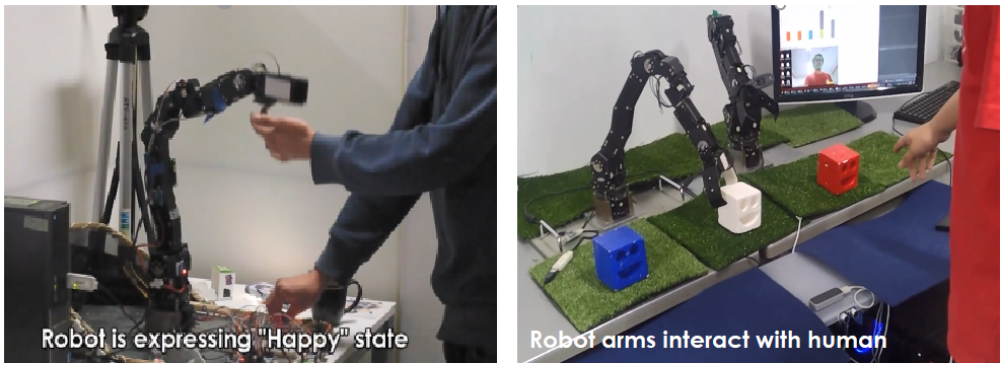


Figure 9.4: Examples of applications preformed using the proposed architectures for development of social intelligent robots

values of *personal*, *social*, and *public*. As the E-FIS, the PAD values are used as outputs of the HI-FIS. However, for the HI-FIS, the human state inputs detected by distance sensors (Kinect, ultrasonic or Intel Real sense) are used in order to change the parameters of the membership functions in the output variables. A total of 3 rules were used in the HI-FIS model.

The outputs of the E-FIS and HI-FIS are represented by the vector $\mathbf{PAD}_{env} = [P_{env}, A_{env}, D_{env}]^T$ and $\mathbf{PAD}_{hi} = [P_{hi}, A_{hi}, D_{hi}]^T$ respectively. The current PAD value is calculated as:

$$\begin{aligned} P_c &= \alpha P_{hi} + \beta P_{env} \\ A_c &= \alpha A_{hi} + \beta A_{env} \\ D_c &= \alpha D_{hi} + \beta D_{env} \end{aligned} \tag{9.1}$$

where the P_c, A_c and D_c indicate the current value of Pleasure Arousal and Dominance respectively. The values of α and β are parameters that can be used to personalize the influence of the the human actions and environmental conditions, and they affect the internal affective model of the robot. For this design, the values were set to α=1 and β=0.5.

The current value of the PAD vector $\mathbf{PAD}_c$ is used to classify the current affect using the K-NN (k-nearest neighbors) algorithm [284]. In the proposed approach, the K-NN algorithm is trained assigning to each affect a vector value inside of the 3-axis PAD space. In K-NN, an unlabeled vector is classified by assigning the label which is most frequent among the k training samples nearest to that query point. Using a value of k=1, a given PAD vector input can be assigned to a class (affect) of it single nearest neighbor.

9.4 Examples of application and discussion

This chapter briefly presents an example of the Human-Robot Interaction research project empowered by NEP. Several demonstrations and versions of the proposed architecture were done using different types of sensory devices. Examples of these applications are shown in figure 9.4. In the left figure, a human change values detected by environmental sensors (temperature, humidity, and brightness) as well as robot sensors (touch and distance) to change the emotional states of the robot. In the right figure, a human select an object using a Leap Motion sensor. Then, robots must decide whether or not to grasp the selected object in base their emotional state. The connection and re-use of sensory, perceptual, and control modules involving these applications were possible due to NEP. Applications performed demonstrates the suitability of NEP for connecting recent MATLAB/Octave versions with other programming languages (i.e. Python and Node.js).

Chapter 10

Human–Robot Interaction at an International Robot Exposition

This chapter briefly explains the software architecture developed using NEP and ROS for enabling HRI between an industrial robot and the visitors of an international robot exposition. Moreover, result and discussion about questionnaires applied to evaluate the proposed application are also presented.

10.1 Software architecture

The software architecture of this application is shown in Figure 10.1. This application required the integration of several sensory devices, computational expensive Deep Learning algorithms, as well as cognitive and robot control modules. Due to the complexity of the system, this architecture was distributed in 3 different computers. Modules in this architecture are mainly connected using the NEP libraries presented in this thesis.

The PC 2 is composed of 4 modules. The module denoted as *Face States* is composed of a set of Python 3 nodes that uses the images obtained from the camera in the head of the robot to obtain the position, emotions and states (i.e., is human interested or distracted) of the closest human to the robot. The *Object Recognition* module uses the cameras in the robot's hands to recognize and localize objects organized in a layout and for manipulation purposes. Information from these two perceptual modules is sent to another *Blackboard* node denoted as *Blackboard Vision*. Images after being processed using OpenCV are sent using a NEP publisher to a GUI designed to display relevant information to users. This GUI was developed using modern web technologies and Javascript libraries, such as Node.js, HTML, CSS, and Vue.js. This module uses output images from the *Face States* module to display the current emotions and states of the human interacting with the robot. Output images obtained from the *Object Recognition* module are used to show which is the current manipulation objective of the robot. The programmed interactive scenario includes some playful activities requiring the Leap Motion information. These two interactive activities are: (i) *hand tracking*, where the robot hand position is remotely controlled by humans; and (ii) *selection of an object*, where a human selects the desired object (a chocolate, a pencil or an eraser) in a shop window by finger pointing. This object is later given by the robot to the human as a gift if a human is detected as happy and interested. Leap Motion information is streamed by the *Blackboard Motion* in PC 1 and read by the other two computers in the network for helping in the decision-making process (in PC 3) and visualize objects selected by humans in the GUI of PC 2. Finally, the PC 3 is a Ubuntu PC with ROS 1.0 installed. The main objective of the nodes in this PC is to perform high-level decision making and low-level robot tasks. On the one hand, decision making is performed using the data obtained from the *Blackboard* systems in PC 1 and PC

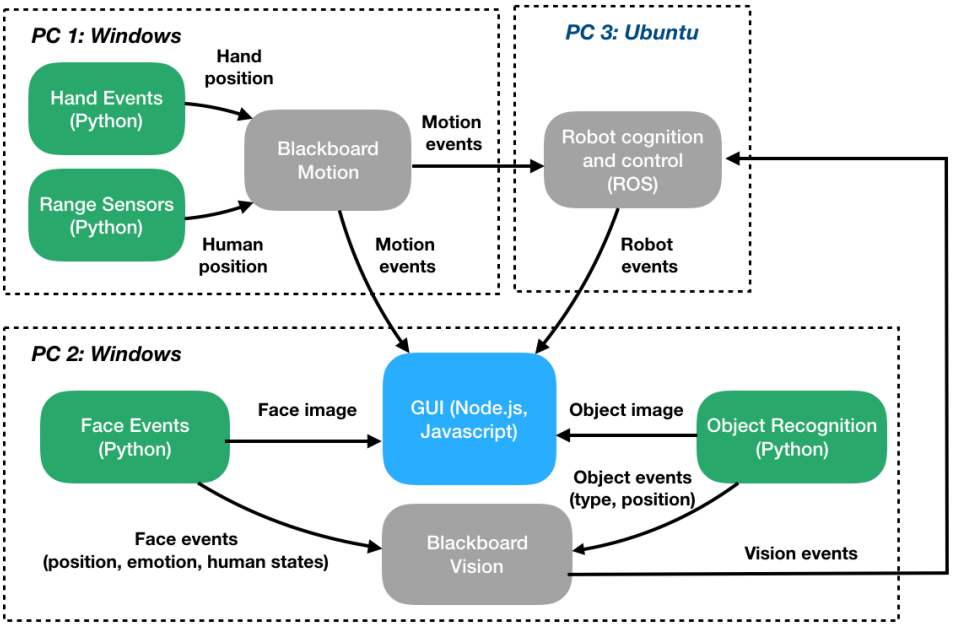


Figure 10.1: General software architecture of the HRI application performed in the International Robot Exhibition (IREX) 2019

Figure 10.2: Example of interaction between a ROS-based robot and humans in a international robot exposition

2. On the other hand, robot movements are performed using ROS libraries for direct and inverse kinematics as well as path planning. Module in PC 3 also sends the interaction and robot status as messages to the GUI in PC 2 to update the elements displayed in the interface. Figure 10.2 shows images of the Open NextStage robot performing pick-and-place and HRI tasks in a real-world setting. This robot using the proposed software architecture developed with NEP and ROS was used for a demonstration in the IREX International Robot Exhibition 2019, which was held in the Tokyo Big Sight, Tokyo, Japan [285].

10.2 Experimental Validation

In order to evaluate the suitability of the proposed HRI application we grasp *Kansei* of visitor using a Semantic difference questionnaire. *Kansei* is an ambiguous term coined in Japan and brought to the west [286]. Like some other popular Japanese terms, such as *manga* and *tsunami*, the word *kansei* is nowadays adopted in many languages. However, this term does not have an exact translation in English [286]. Closest meanings include sensitivity, affection,

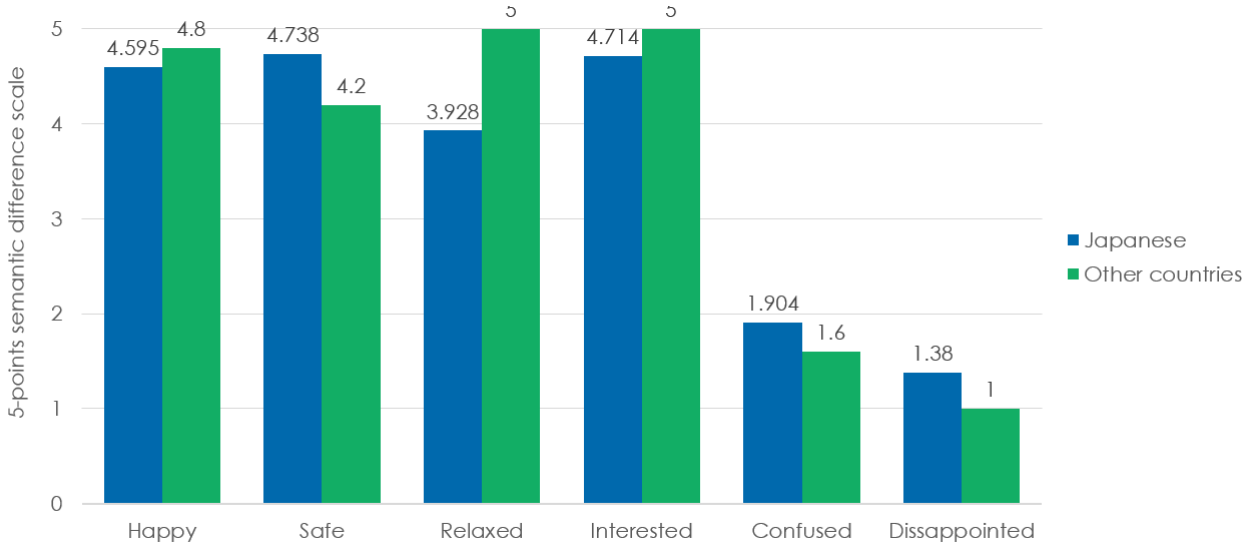


Figure 10.3: People feeling about the HRI scenario

aesthetics, emotion, want, need, and feeling [286, 287]. A more complete definition is expressed in [287] as *"the feeling felt by the receiver of stimuli contained in the atmosphere of a situation"*. This reaction to stimuli (*kansei*) is highly influenced by the past experiences of users and a combination of sensory modalities (especially eyesight) [287].

Kansei Engineering is a consumer-oriented technology [288] and suitable *human-centered* design and ergonomic approach for the development of products and services. This research area was founded in 1970 by Mitsuo Nagamachi and its applicability has been expanded beyond design and manufacturing areas [110]. Historically, this research area has been widely explored in Asian countries such as South Korea, Japan, China, and Malaysia. However, it has also gained popularity in some occidental countries [289]. The main goal of *Kansei Engineering* is to improve the quality of life, comfort and enjoyment of people [286] by the development of novel emotion-based products using a *human-oriented* (i.e., oriented to the human mind) approach [287]. For this, is keystone to understand and grasp the human *kansei*, which is done by sensing facial expressions, eyes, spoken works and other human reactions [289]. This information can be analyzed using psychological, ergonomic and engineering methods for its posterior translation into design specifications. These specifications or design items can be integrated in the final product to: (i) stimulate the user's emotions or *kansei*; and (ii) meet needs and implicit expectations of consumers [286, 287, 290].

A key step in *Kansei Engineering Type I* (*KE Type I*), which methodology is described in [291, 292, 293] is to identify a set of relevant *Kansei* words able to describe subjective feelings about the product. Few examples are: *modern*, *elegant*, *old*, *dynamic*, *cute*, *happy* and *angry*. The selection of these words is often done from a survey in state-of-art articles or by consulting experts in the area where the product will be applied. These words are then arranged in a semantic differential (SD) scale [294] to enable data collection in experimental sessions. Experimental sessions in this type of *Kansei Engineering* basic methodology often imply: (i) to present of the product to users; and (ii) to grasp the user's feelings, impressions or reactions about each sample of the products presented. This data is obtained using the corresponding SD scale sheet (where the *Kansei* words previously selected for the experimental session are displayed). Results and interpretations from questionnaires can be used to validate the feasibility of some product or service or help in the creation of new emotional products.

For this project KE Type I was used to validate the feasibility the proposed HRI scenario by grasping emotional reactions of visitors. *Kensei* word were selected after analysing state-of-art

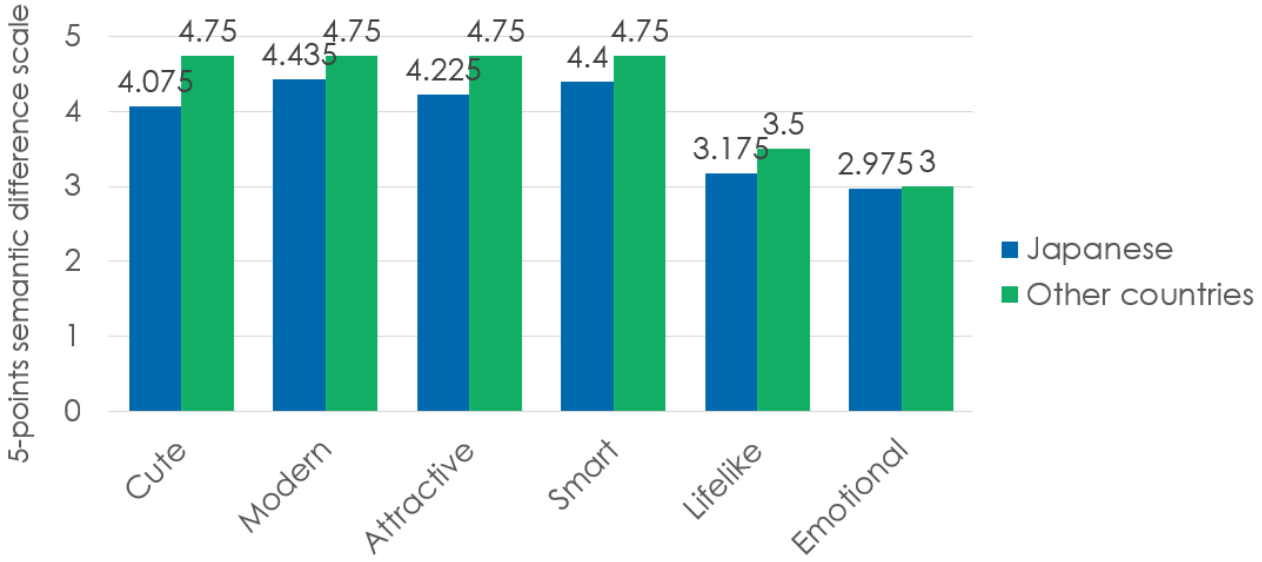


Figure 10.4: People feeling about the robot's design intelligence and anthropomorphism

works in HRI. These words are grouped in two categories: (1) people feeling when interacting with the robot (2) people feeling about the design and behavior of the robots. Words involved in group 1 are: *happy/unhappy*, *safe/danger*, *relaxed/anxious*, *interested/boring*, *confused/clear*, *disappointed/amused*. Words involved in group 2 are: *cute/ugly*, *modern/old*, *attractive/unattractive*, *smart/stupid*, *lifelike/unreal*, *emotional/emotionless*. Results from questionnaires are shown in figures 10.3 and 10.4.

10.3 Discussion of results

Results from *kansei* questionnaires show that in general people had very positive feelings when interacting with the robot. Feeling such as *happy*, *safe*, and *interested* got very high scores. As shown in figure 10.3 the actions of the robot were in general clear and most of the visitors were amused about the application. Figure 10.4 shows that people considers the design of the Open-Next Stage robot as *cute*, *modern* and *attractive*. Even when people consider robot behaviors as *smart*, visitors still consider this robot as a machine and not a living creature with few emotional intelligence capabilities. This can be due to the lack of emotional expressivity in the robot motions.

Chapter 11

Conclusions, Limitations and Future Work

This chapter summarizes, discusses and concludes the research findings and contributions of this thesis, which are grounded by the objectives and research questions defined in chapter 1.

11.1 Discussion summary

Chapter 2 focused to ask research question RQ1. The findings of this chapter indicate that there is a need for a more user-friendly distributed robotics framework. Therefore, the first objective of this thesis was to propose a novel and user-friendly distributed robotics framework, which referred to in this thesis as NEP. Chapter 3 briefly described the main libraries and frameworks used in the development of the software tools proposed in this thesis. Chapter 4 described the main libraries created and features of NEP, which development is done for satisfy objective *O1* (defined in chapter 1). Findings in this chapter indicate that NEP can be used not only to create usable software artifacts for robotics in *human-centered* research tasks and applications, but also as a low latency option for academic-oriented projects; therefore, answering RQ7. Chapter 5 focused on provides a wide vision of current VPEs for EUD of social and service robots for adult and nonskilled end-users in every-day and social research scenarios rather than industrial settings. This chapter was used to ask research questions RQ2, RQ3, RQ4, and RQ5. For this, a systematic review was performed. The analysis is done in this chapter also includes an early version of RIZE, the EUD tool developed in this thesis for meet objective *O2*. This chapter solved RQ2, RQ3 and RQ4 by presenting a in depth overview of VPE tools for the development of social and service robots. Special attention was given in the analysis of technologies and AI modelling methods used in this tools. The findings of this chapter indicate that there is a need for more accessible, adaptable, modular, extendable and flexible tools and technologies to support and enable end users to become end-user developers of their system. This chapter also highlights the inherent complexity of most distributed robotics frameworks. This produces some accessibility and usability barriers that make it difficult to create EUD tools promoting independence between end users and high-tech scribes. A set of challenges were also identified. These are related to accessibility to external devices and resources, modularity of the human-robot interaction primitives, scalability when large programs are needed, level of abstraction, benchmarking, explainability and control of the resulting robot behaviors, support for distributed robot frameworks, as well as simulation and debugging. The final version of RIZE was the focus in overcoming some of the most critical challenges identified. Features of the final version of RIZE are described in chapter 6. This chapter also presented usability and UX guidelines, design approaches and final software architecture of RIZE. This final version is the result of several iterations after getting feedback from real end-users were used the different versions to create their own applications. These applications were executed and tested "in the

wild" scenarios; therefore, satisfying objective *O3*. I realized that the successfully performing of this type of experiment is much more complex than those only implemented in laboratory settings where most factors are ideal and controlled. However, experiments "in the wild" also provides much richer insights form the technical and usability point of view. Many academic and end-user in projects HRI, CRI, affective computing, industrial robots, and biomechanics were supported using NEP and RIZE. However, this thesis presented the 3 most relevant, which mainly influenced in the improvement of the proposed framework. These projects were presented chronologically in chapters 7, 8, 9 and 10. The project of chapter 7, was the first major challenge addressed. Even when the proposed frameworks were in a very early stage of development, end-users were able to design and execute their desired application. Usability issues when installing and executing RIZE in this project indicated a need to base this tool in more advanced software libraries that enable the creation of more accessible and user-friendly software. Therefore a shift from Flask to Node.js, Vue.js, and electron was performed. This approach enables end-users to install software with the simplicity of commercial and usable oriented software products. Several applications and projects of CRI were also supported in this doctoral work. These were presented in chapter 8. Examples presented in this chapter proved the suitability of RIZE for enabling the creation and execution of CRI applications performed "in the wild" (answering RQ8). Feedback from end-users and presented technical issues in pilot tests was used to improve the final software architecture of RIZE. Finally research questions RQ6 and RQ8 are answered in chapters 9 and 10. An advanced HRI system is presented in this chapter. This robotics system integrates many robotics components written in different programming languages and executed in different computers. Moreover, feedback obtained in questionnaires from visitors validates the suitability of the interactive scenario developed.

11.2 Limitations and future work

This section presents the current limitations of the presented research as well as how this limitation can be addressed in future research projects.

Integration of RIZE with simulators for robotics

Chapter 8 prove the applicability and suitability of RIZE, enabling end users to create their own HRI applications. The development, debugging, and execution of these applications was directly applied by end users in the real robots. In some cases, end users can require to use a virtual robot before implementing their application in the real robot. As described in chapter 5 one of the most relevant challenges identified in VPEs for EUD of robotics is the availability or interoperability with user-friendly simulators for enabling offline programming of robots. However, the creation of a simulation environment is, in fact, a complex and time-consuming task. Therefore, this activity can be addressed in future research projects. Two alternatives enabling the creation of user-friendly and cross-platform 3D environments are Unity (a popular game engine) and three.js (a Javascript library for the creation of 3D scenarios). These two 3D modeling and simulation tools can be easily communicated with RIZE using the Javascript and C# versions of NEP.

Hosting RIZE as a website

RIZE is a web-application developed in node.js (which enables the execution of Javascript code outside the browser) and electron (described in chapter 3). This approach enables the installation and execution of RIZE as any other native desktop application. With few modifications is possible to host RIZE in a local or web server as any web-page. This can enable the use

of RIZE from smart-devices or any other device with a web browser. However, POSIX and ZeroMQ sockets in Javascript only work in environments executed on top of node.js. To communicate RIZE interface with a local or web server the use HTTP requests, Websockets or Server-Sent-Events is required. Future projects will include the creation of local servers for ROS users. Publication of RIZE as a web-page accessible via the internet is also a suitable option to explore.

Creation of a bridge server for NEP and ROS

In the proposed approach code developed with NEP can be reused in ROS as JSON messages sent in strings. Users of NEP only require to change a flag in the definition of the nodes to select the desired communication option (ROS, ROS 2.0, and ZeroMQ). However, many academic projects using ROS can require the use of ROS standard messages for performing low-level tasks. The creation of a proxy/bridge module able to directly communicate messages from ZeroMQ to ROS, ZeroMQ to ROS 2.0 or ROS and ROS 2.0 using standard ROS message can be a relevant contribution to the ROS community. This bridge server can be integrated into the NEP Discovery Service Master Node (described in 4).

Exploration of novel End-User Programming approaches

The RIZE framework has been designed to be modular. Therefore, the EUP approach supported by RIZE can be easily extended, modified, or substituted. In this context, two projects are left as future work: (i) to substitute Google Blockly by an authoring programming environment developed in Javascript for enabling more flexibility, robustness as well as the integration of more advanced HRI aspects, such as emotions; (ii) to extend RIZE using kinesthetic teaching. Motion skills learned using this approach can be saved and reproduced in RIZE as new animations.

Integration of emotions and a utility system

By using a more flexible alternative to Google Blockly it can be possible to explore novel programming alternatives enabling end user to generate robots with advanced emotional intelligence. Rather than be managed by static priorities, robot behaviors modeled as BTs can be controlled in the highest cognitive level by a utility system taking into account the robot's emotions, personality, desires, goals, and other HRI factors. This approach can be used to generate more believable social intelligent robots able to display more engaged and dynamic behaviors that adapt to the current HRI scenario. Unlike end-to-end and black-box decision-making approaches, this approach can provide a good trade-off between performance and explainability. This is due that robot behaviors will be always explained as BTs. These BTs can also be learned from interactions. Moreover, emotional values can be used to affect the trajectories or control parameters in those modules executing the motions of the robot, such as presented in [224, 295].

11.3 Conclusions

This thesis presented two robotics software frameworks (NEP and RIZE) aimed to improve the usability and flexibility of robotics systems. On the one hand, NEP is a distributed robotic framework constituted by a set of libraries in Python, Java, C#, and JavaScript. NEP enables the integration of sensory, perceptual, cognitive and control systems for robotics as well as the creation of cross-platform and user-friendly applications. We prove the technological suitability of NEP briefly describing the general software architectures of several real-world applications

where robots require to interact with humans in "*the wild*" settings rather than laboratories. NEP can be used as a glue software enabling the integration of state-of-art robotics research frameworks and modules, with *human-centered* software interfaces. NEP have also been designed to be easy-to-implemented by the application programmer and easy-to-install by the *end user*. The presented study proved technically superiority (lower latency) of the proposed approach in both local-host and LAN communication against the state-of-art approach for communicating robotics written in programming languages and launched form operating systems not fully supported by ROS. Findings indicate that NEP can be used not only to create usable software artifacts for robotics in *human-centered* research tasks and applications, but also as a low latency option for *academic-oriented* projects connecting ROS with non-ROS enabled modules or devices. Moreover, ZeroMQ and nanomsg sockets are present in almost all programming languages and devices such as Android and iOS smartphones. Therefore, supporting a new programming language becomes a trivial issue. Current NEP libraries are accessible to developers as free to use and open source in [129]. On the other hand, RIZE is a EUD solution. RIZE is embedded in a is a web-based interface that is executed a desktop application. This approach is proposed to support HRI "in the wild", were have a stable and robust internet connection is in many cases not possible. The evolution of software architecture and applications were presented. These interactions were updated in the base of the need of real end users. The main contributions of this thesis are described bellow:

- A systematically identification and analysis of relevant VPEs for enabling the EUD paradigm and creation with social and service robots
- An in-depth understanding of the behavior modeling approaches currently used by these EUD tools for enabling the creation of social intelligent robots
- Development and evaluation of a set of inter-process communication libraries for supporting the design of complex software architectures for robotics composed of modules written in modern programming languages and executed in different OS
- Development of a modular and reactive behavior modelling and execution engine for design intelligent robotics systems
- Design and evaluation of a novel VPE using modern web technologies and good practices for development of robotics applications.
- Development and validation of a several distributed control architecture enabling the social robots to perform in HRI "in the wild" applications.

Publications

The publications listed below were produced and accepted during this doctoral work.

- **Coronado, E.**, Mastrogiovanni, F. Indurkhya, B. , Venture, G. (2020). *Visual programming environments for end-user development of intelligent and social robots: a systematic review.* Journal of Computer Languages (Accepted).
- **Coronado, E.**, Venture, G. Yamanobe N. (2020). *Applying Kansei/Affective Engineering Methodologies in the Design of Social and Service Robots: A Systematic Review.* International Journal of Social Robotics (Accepted).
- **Coronado, E.**, Rincon, L., Venture, G. (2020). *Connecting MATLAB/Octave to perceptual, cognitive and control components for the development of intelligent robotic systems.* In ROMANSY 23–Robot Design, Dynamics and Control (Accepted).
- **Coronado, E.**, Venture, G. (2020). *Towards IoT-Aided Human–Robot Interaction Using NEP and ROS: A Platform-Independent, Accessible and Distributed Approach.* Sensors 2020, 20, 1500.
- **Coronado, E.**, Indurkhya, X., Venture, G. (2019). *Robots Meet Children, Development of Semi-Autonomous Control Systems for Children-Robot Interaction in the Wild.* In 2019 IEEE 4th International Conference on Advanced Robotics and Mechatronics (ICARM) (pp. 360-365). IEEE.
- **Coronado, E.**, Mastrogiovanni, F., Venture, G. (2018). *Development of intelligent behaviors for social robots via user-friendly and modular programming tools.* In 2018 IEEE Workshop on Advanced Robotics and its Social Impacts (ARSO) (pp. 62-68). IEEE.
- **Coronado, E.**, Mastrogiovanni, F., Venture, G. (2018). *Design of a human-centered robot framework for end-user programming and applications.* In ROMANSY 22–Robot Design, Dynamics and Control (pp. 450-457). Springer, Cham.

Other publications produced and using the developed software tools for supporting different HRI research activities are:

- Rida, F., Rincon, L., **Coronado, E.**, Nait-ali, A., Venture, G. (2020). *From Motion to Emotion Prediction: A Hidden Biometrics Approach.* In Hidden Biometrics (pp. 185-202). Springer, Singapore.
- Rincon L., Fillol F., **Coronado E.**, Shi Y., Chen Y., Bourguet M. Venture G. (2019). *Adaptive Optimal Predictive Control System For Cognitive Manipulator Robots Based On Human Engagement/Intention And Deep Dynamic Perception*, 25 Japan IFToMM Conference, pp. 23-30, Tokyo, Japan.

- Pattar, S. P., **Coronado, E.**, Rincon, L. R., Venture, G. (2019). *Intention and Engagement Recognition for Personalized Human-Robot Interaction, an integrated and Deep Learning approach.* In 2019 IEEE 4th International Conference on Advanced Robotics and Mechatronics (ICARM) (pp. 93-98). IEEE.

- Rincon, L., **Coronado, E.**, Law, C., Venture, G. (2019). *Adaptive cognitive robot using dynamic perception with fast deep-learning and adaptive on-line predictive control.* In IFToMM World Congress on Mechanism and Machine Science (pp. 2429-2438). Springer, Cham.

- Indurkhya X., **Coronado E.**, Izui T. , Zguda P., Indurkhya B., Venture G. (2019), *Creating a robust vocalization-based protocol for analyzing CRI group studies in the wild*, ACM/IEEE International Conference on Human Robot Interaction, Daegu, Korea.

- Rincon, L., **Coronado, E.**, Hendra, H., Phan, J., Zainalkefli, Z., Venture, G. (2019). *Adaptive Fuzzy and Predictive Controllers for Expressive Robot Arm Movement during Human and Environment Interaction.* International Journal of Mechanical Engineering and Robotics Research, 8(2).

- Rincon, L., **Coronado, E.**, Hendra, H., Phan, J., Zainalkefli, Z., Venture, G. (2018). *Expressive states with a robot arm using adaptive fuzzy and robust predictive controllers.* In 2018 3rd International Conference on Control and Robotics Engineering (ICCRE) (pp. 11-15). IEEE.

Results from publications show the technological feasibility of NEP and RIZE for the support of advanced and interdisciplinary and research activities in robotics.

Appendices

Appendix A

Tables

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

Table A.1: Dimensions used for answer RQ4

Appendix B

Code examples

B.1 nep.js package

```
{
  "name": "nep-js",
  "version": "0.1.1",
  "description": "",
  "main": "nep/nep.js",
  "umd:main": "nep/nep.umd.js",
  "module": "nep/nep.mjs",
  "source": "src/nep.js",
  "scripts": {
    "test": "jest",
    "build": "microbundle",
    "prepublish": "npm run build"
  },
  "keywords": [],
  "author": "Enrique Coronado <enriquecoronadozu@gmail.com> (https
   ://enriquecoronadozu.github.io/NEP/)",
  "license": "ISC",
  "dependencies": {
    "zeromq": "^6.0.0-beta.6"
  },
  "devDependencies": {
    "@babel/preset-env": "^7.4.5",
    "babel-jest": "^24.8.0",
    "eslint": "^5.16.0",
    "eslint-config-prettier": "^4.3.0",
    "eslint-plugin-prettier": "^3.1.0",
    "husky": "^2.4.1",
    "jest": "^24.8.0",
    "lint-staged": "^8.2.1",
    "microbundle": "^0.11.0",
    "prettier": "^1.18.2"
  },
  "husky": {
    "hooks": {
      "pre-commit": "npm test"
    }
  },
  "files": [
    "nep",
    "index.d.ts",
    "package.json"
  ]
}
```

Listing B.1: Package configuration file for nep.js library

B.2 RIZE package

```
{
  "name": "rize_social",
  "version": "0.0.1",
  "productName": "RIZE Social",
  "description": "Robot Interface From Zero Experience",
  "main": "main.js",
  "scripts": {
    "postinstall": "install-app-deps",
    "start": "npm install && electron .",
    "pack": "build --dir",
    "dist": "build"
  },
  "build": {
    "appId": "RIZE Social",
    "extraResources": "python_scripts",
    "dmg": {
      "contents": [
        {
          "x": 110,
          "y": 150
        },
        {
          "x": 240,
          "y": 150,
          "type": "link",
          "path": "/Applications"
        }
      ]
    },
    "linux": {
      "target": [
        "AppImage",
        "deb"
      ]
    },
    "win": {
      "target": "NSIS",
      "icon": "images/rize2.png"
    }
  },
  "author": "Enrique Coronado",
  "license": "ISC",
  "dependencies": {
    "axios": "^0.19.0",
    "fix-path": "^2.1.0",
    "jquery": "^3.4.1",
    "nep-js": "0.0.8",
    "vue": "^2.6.11",
    "vuetify": "2.2.1",
    "zeromq": "^6.0.0-beta.6"
  },
  "devDependencies": {
    "electron-builder": "^20.44.2",
    "electron": "^7.1.7"
  }
}
```

Listing B.2: Package configuration file for RIZE

Bibliography

[1] https://vuejs.org/, 2020.

[2] P. Harwood, "Multi modal human robot interaction interface," Master's thesis, Ecole Central of Nantes, 2016, 6 2016.

[3] D. Mark, "Ai architectures: A culinary guide," *Game Developer Magazine*, vol. 19, no. 8, pp. 7–12, 2012.

[4] L. R. Ardila, E. Coronado, H. Hendra, J. Phan, Z. Zainalkefli, and G. Venture, "Adaptive fuzzy and predictive controllers for expressive robot arm movement during human and environment interaction," *International Journal of Mechanical Engineering and Robotics Research*, vol. 8, no. 2, 2019.

[5] 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*. Springer, 2019, pp. 151–163.

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

[7] L. Pu, W. Moyle, C. Jones, and M. Todorovic, "The effectiveness of social robots for older adults: a systematic review and meta-analysis of randomized controlled studies," *The Gerontologist*, vol. 59, no. 1, pp. e37–e51, 2019.

[8] E. Coronado, J. Villalobos, B. Bruno, and F. Mastrogiovanni, "Gesture-based robot control: Design challenges and evaluation with humans," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, May 2017, pp. 2761–2767.

[9] M. A. Goodrich and A. C. Schultz, "Human-robot interaction: a survey," *Foundations and Trends in Human-Computer Interaction*, vol. 1, no. 3, pp. 203–275, 2007.

[10] J. A. Adams and M. Skubic, "Introduction to the special issue on human–robot interaction," *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, vol. 35, no. 4, pp. 433–437, 2005.

[11] D. Mariette and B. S. Meyerson, "Top 10 Emerging Technologies 2019," World Economic Forum, Tech. Rep., 06 2019.

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

[13] J. Lindblom and R. Andreasson, "Current challenges for ux evaluation of human-robot interaction," in *Advances in ergonomics of manufacturing: Managing the enterprise of the future*. Springer, 2016, pp. 267–277.

[14] C. Lutz, M. Schöttler, and C. P. Hoffmann, "The privacy implications of social robots: Scoping review and expert interviews," *Mobile Media & Communication*, vol. 7, no. 3, pp. 412–434, 2019.

[15] E. Fosch-Villaronga, C. Lutz, and A. Tamò-Larrieux, "Gathering expert opinions for social robots' ethical, legal, and societal concerns: Findings from four international workshops," *International Journal of Social Robotics*, pp. 1–18, 2019.

[16] M. F. Lohmann, "Liability issues concerning self-driving vehicles," *European Journal of Risk Regulation*, vol. 7, no. 2, pp. 335–340, 2016.

[17] K. Yogeeswaran, J. Złotowski, M. Livingstone, C. Bartneck, H. Sumioka, and H. Ishiguro, "The interactive effects of robot anthropomorphism and robot ability on perceived threat and support for robotics research," *Journal of Human-Robot Interaction*, vol. 5, no. 2, pp. 29–47, 2016.

[18] J. Smids, S. Nyholm, and H. Berkers, "Robots in the workplace: a threat to—or opportunity for—meaningful work?" *Philosophy & Technology*, pp. 1–20, 2019.

[19] D. Brugali and A. Shakhimardanov, "Component-based robotic engineering (part ii)," *IEEE Robotics & Automation Magazine*, vol. 17, no. 1, pp. 100–112, 2010.

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

[21] 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*. Springer, 2019, pp. 450–457.

[22] P. Simoens, M. Dragone, and A. Saffiotti, "The internet of robotic things: A review of concept, added value and applications," *International Journal of Advanced Robotic Systems*, vol. 15, no. 1, p. 1729881418759424, 2018.

[23] E. Tsardoulias and P. Mitkas, "Robotic frameworks, architectures and middleware comparison," *arXiv preprint arXiv:1711.06842*, 2017.

[24] L. Joseph and J. Cacace, *Mastering ROS for Robotics Programming: Design, build, and simulate complex robots using the Robot Operating System*. Packt Publishing Ltd, 2018.

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

[26] D. F. Glas, T. Kanda, and H. Ishiguro, "Human-robot interaction design using interaction composer eight years of lessons learned," in *2016 11th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, March 2016, pp. 303–310.

[27] H.-L. Cao, G. Van de Perre, J. Kennedy, E. Senft, P. G. Esteban, A. De Beir, R. Simut, T. Belpaeme, D. Lefeber, and B. Vanderborght, "A personalized and platform-independent behavior control system for social robots in therapy: development and applications," *IEEE Transactions on Cognitive and Developmental Systems*, 2018.

[28] P. G. Esteban, P. Baxter, T. Belpaeme, E. Billing, H. Cai, H.-L. Cao, M. Coeckelbergh, C. Costescu, D. David, A. De Beir *et al.*, "How to build a supervised autonomous system for robot-enhanced therapy for children with autism spectrum disorder," *Paladyn, Journal of Behavioral Robotics*, vol. 8, no. 1, pp. 18–38, 2017.

[29] M. Coeckelbergh, C. Pop, R. Simut, A. Peca, S. Pintea, D. David, and B. Vanderborght, "A survey of expectations about the role of robots in robot-assisted therapy for children with asd: Ethical acceptability, trust, sociability, appearance, and attachment," *Science and engineering ethics*, vol. 22, no. 1, pp. 47–65, 2016.

[30] G. N. Yannakakis and J. Togelius, *Artificial intelligence and games*. Springer, 2018, vol. 2.

[31] A. Fiske, P. Henningsen, and A. Buyx, "Your robot therapist will see you now: Ethical implications of embodied artificial intelligence in psychiatry, psychology, and psychotherapy," *Journal of medical Internet research*, vol. 21, no. 5, p. e13216, 2019.

[32] F. E. Ritter, G. D. Baxter, and E. F. Churchill, "User-centered systems design: a brief history," in *Foundations for designing user-centered systems*. Springer, 2014, pp. 33–54.

[33] 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 *Proceedings of the 31st Australian Conference on Human-Computer-Interaction*, 2019, pp. 30–41.

[34] A. D. Frederiks, J. R. Octavia, C. Vandevelde, and J. Saldien, "Towards participatory design of social robots," in *IFIP Conference on Human-Computer Interaction*. Springer, 2019, pp. 527–535.

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

[36] G. Fischer, D. Fogli, and A. Piccinno, "Revisiting and broadening the meta-design framework for end-user development," in *New perspectives in end-user development*. Springer, 2017, pp. 61–97.

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

[38] 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," *Journal of Systems and Software*, vol. 149, pp. 101–137, 2019.

[39] G. Fischer, E. Giaccardi, Y. Ye, A. G. Sutcliffe, and N. Mehandjiev, "Meta-design: a manifesto for end-user development," *Communications of the ACM*, vol. 47, no. 9, pp. 33–37, 2004.

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

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

[42] A. J. Ko, R. Abraham, L. Beckwith, A. Blackwell, M. Burnett, M. Erwig, C. Scaffidi, J. Lawrance, H. Lieberman, B. Myers *et al.*, "The state of the art in end-user software engineering," *ACM Computing Surveys (CSUR)*, vol. 43, no. 3, pp. 1–44, 2011.

[43] Z. Zhu and H. Hu, "Robot learning from demonstration in robotic assembly: A survey," *Robotics*, vol. 7, no. 2, p. 17, 2018.

[44] J. Huang and M. Cakmak, "Programming by demonstration with user-specified perceptual landmarks," *arXiv preprint arXiv:1612.00565*, 2016.

[45] C. Ding, J. Wu, Z. Xiong, and C. Liu, "A reconfigurable pick-place system under robot operating system," in *International Conference on Intelligent Robotics and Applications*. Springer, 2018, pp. 437–448.

[46] K. Fischer, F. Kirstein, L. C. Jensen, N. Krüger, K. Kukliński, M. V. aus der Wieschen, and T. R. Savarimuthu, "A comparison of types of robot control for programming by demonstration," in *2016 11th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*. IEEE, 2016, pp. 213–220.

[47] D.-Q. Zhang and K. Zhang, "On the design of a generic visual programming environment," in *Proceedings. 1998 IEEE Symposium on Visual Languages*. IEEE, Sep. 1998, pp. 88–89.

[48] P. P. Ray, "A survey on visual programming languages in internet of things," *Scientific Programming*, vol. 2017, 2017.

[49] M. Jung and P. Hinds, "Robots in the wild: A time for more robust theories of human-robot interaction," 2018.

[50] ibug. (2016) Special issue on behavior analysis "in-the-wild". [Online]. Available: http://ibug.doc.ic.ac.uk/resources/SI-HBAW/

[51] S. Sabanovic, M. P. Michalowski, and R. Simmons, "Robots in the wild: Observing human-robot social interaction outside the lab," in *9th IEEE International Workshop on Advanced Motion Control, 2006.* IEEE, 2006, pp. 596–601.

[52] F. Alaieri and A. Vellino, "Ethical decision making in robots: Autonomy, trust and responsibility," in *International conference on social robotics*. Springer, 2016, pp. 159–168.

[53] T. Tangiuchi, D. Mochihashi, T. Nagai, S. Uchida, N. Inoue, I. Kobayashi, T. Nakamura, Y. Hagiwara, N. Iwahashi, and T. Inamura, "Survey on frontiers of language and robotics," *Advanced Robotics*, vol. 0, no. 0, pp. 1–31, 2019.

[54] T. Kanda and H. Ishiguro, *Human-robot interaction in social robotics*. CRC Press, 2016.

[55] M. Mast, M. Burmester, B. Graf, F. Weisshardt, G. Arbeiter, M. Španěl, Z. Materna, P. Smrž, and G. Kronreif, "Design of the human-robot interaction for a semi-autonomous service robot to assist elderly people," in *Ambient Assisted Living*. Springer, 2015, pp. 15–29.

[56] C. Clabaugh and M. Matarić, "Escaping oz: Autonomy in socially assistive robotics," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 2, pp. 33–61, 2019.

[57] A. Zaraki, L. Wood, B. Robins, and K. Dautenhahn, "Development of a semi-autonomous robotic system to assist children with autism in developing visual perspective taking skills," in *2018 27th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*. IEEE, 2018, pp. 969–976.

[58] M. van Steen and A. S. Tanenbaum, "A brief introduction to distributed systems," *Computing*, vol. 98, no. 10, pp. 967–1009, 2016.

[59] M. Hailperin, *Operating Systems and Middleware: Supporting Controlled Interaction*. Max Hailperin, 2007.

[60] *Transport and Middleware Layers*. Boston, MA: Springer US, 2007, pp. 65–75. [Online]. Available: https://doi.org/10.1007/978-0-387-39023-9_5

[61] T. Noergaard, *Embedded systems architecture: a comprehensive guide for engineers and programmers*. Newnes, 2012.

[62] A. Shakhimardanov, N. Hochgeschwender, M. Reckhaus, and G. K. Kraetzschmar, "Analysis of software connectors in robotics," in *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*. IEEE, 2011, pp. 1030–1035.

[63] R. Otap, "Development of a robotic testbed infrastructure with dynamic service discovery," 2013.

[64] http://pypl.github.io/PYPL.html, 2020.

[65] U. Bhandari, T. Neben, K. Chang, and W. Y. Chua, "Effects of interface design factors on affective responses and quality evaluations in mobile applications," *Computers in Human Behavior*, vol. 72, pp. 525–534, 2017.

[66] M. Thüring and S. Mahlke, "Usability, aesthetics and emotions in human–technology interaction," *International journal of psychology*, vol. 42, no. 4, pp. 253–264, 2007.

[67] 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 *ICRA workshop on open source software*, vol. 3, no. 3.2. Kobe, 2009, p. 5.

[68] https://design.ros2.org/articles/why_ros2.html, 2019.

[69] http://wiki.ros.org/ROS/Introduction, 2019.

[70] R. Toris, J. Kammerl, D. V. Lu, J. Lee, O. C. Jenkins, S. Osentoski, M. Wills, and S. Chernova, "Robot web tools: Efficient messaging for cloud robotics," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2015, pp. 4530–4537.

[71] http://wiki.ros.org/melodic, 2020.

[72] https://www.ros.org/reps/rep-0003.html, 2020.

[73] http://www.tivipe.com/TVPeducation/TVPuse.pdf, 2019.

[74] E. Barakova, J. Gillesen, B. Huskens, and T. Lourens, "End-user programming architecture facilitates the uptake of robots in social therapies," *Robotics and Autonomous Systems*, vol. 61, no. 7, pp. 704 – 713, 2013.

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

[76] http://wiki.ros.org/rosbridge_suite, 2019.

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

[78] C. Ciliberto, "Connecting yarp to the web with yarp. js," *Frontiers in Robotics and AI*, vol. 4, p. 67, 2017.

[79] C. S. V. Gutiérrez, L. U. S. Juan, I. Z. Ugarte, and V. M. Vilches, "Towards a distributed and real-time framework for robots: Evaluation of ros 2.0 communications for real-time robotic applications," *arXiv preprint arXiv:1809.02595*, 2018.

[80] https://www.ros.org/reps/rep-2000.html, 2020.

[81] T. Sumalan, E. Lupu, and R. Arsinte, "Real time operating system options in connected embedded equipment for distributed data acquisition," *Carpathian Journal of Electronic and Computer Engineering*, vol. 11, no. 2, pp. 35–58, 2018.

[82] https://index.ros.org/doc/ros2/Tutorials/Real-Time-Programming/, 2020.

[83] I. C. Bertolotti and G. Manduchi, *Real-time embedded systems: open-source operating systems perspective*. CRC press, 2017.

[84] G. Tong and C. Liu, "Supporting soft real-time sporadic task systems on uniform heterogeneous multiprocessors with no utilization loss," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 9, pp. 2740–2752, 2015.

[85] G. Metta, P. Fitzpatrick, and L. Natale, "Yarp: yet another robot platform," *International Journal of Advanced Robotic Systems*, vol. 3, no. 1, p. 8, 2006.

[86] G. Metta, G. Sandini, D. Vernon, L. Natale, and F. Nori, "The icub humanoid robot: an open platform for research in embodied cognition," in *Proceedings of the 8th workshop on performance metrics for intelligent systems*. ACM, 2008, pp. 50–56.

[87] D. C. Schmidt, "The adaptive communication environment: An object-oriented network programming toolkit for developing communication software," 1993.

[88] S.-G. Chitic, J. Ponge, and O. Simonin, "Are middlewares ready for multi-robots systems?" in *International Conference on Simulation, Modeling, and Programming for Autonomous Robots*. Springer, 2014, pp. 279–290.

[89] http://wiki.ros.org/catkin, 2020.

[90] N. Ando, T. Suehiro, K. Kitagaki, T. Kotoku, and W.-K. Yoon, "Rt-middleware: distributed component middleware for rt (robot technology)," in *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2005, pp. 3933–3938.

[91] G. Magyar, P. Sinčák, and Z. Krizsán, "Comparison study of robotic middleware for robotic applications," in *Emergent Trends in Robotics and Intelligent Systems*. Springer, 2015, pp. 121–128.

[92] https://rfc.zeromq.org/spec/15/, 2019.

[93] http://https://nanomsg.org/, 2018.

[94] A. Dworak, F. Ehm, P. Charrue, and W. Sliwinski, "The new cern controls middleware," in *Journal of Physics: Conference Series*, vol. 396, no. 1. IOP Publishing, 2012, p. 012017.

[95] M. Al-Turany, P. Buncic, P. Hristov, T. Kollegger, C. Kouzinopoulos, A. Lebedev, V. Lindenstruth, A. Manafov, M. Richter, A. Rybalchenko *et al.*, "Alfa: The new alice-fair software framework," in *Journal of Physics: Conference Series*, vol. 664, no. 7. IOP Publishing, 2015, p. 072001.

[96] L. Mirabito, "Zdaq, a light data acquisition framework based on zeromq," *Journal of Instrumentation*, vol. 14, no. 10, p. C10007, 2019.

[97] E. Babaians, M. Tamiz, Y. Sarti, A. Mogoei, and E. Mehrabi, "Ros2unity3d; high-performance plugin to interface ros with unity3d engine," in *2018 9th Conference on Artificial Intelligence and Robotics and 2nd Asia-Pacific International Symposium*. IEEE, 2018, pp. 59–64.

[98] https://nodejs.org/en/, 2020.

[99] C. Anderson, "The model-view-viewmodel (mvvm) design pattern," in *Pro Business Applications with Silverlight 5*. Springer, 2012, pp. 461–499.

[100] M. Street, A. Passaglia, and P. Halliday, *Complete Vue. js 2 web development: practical guide to building end-to-end web development solutions with Vue. js 2*. Packt Publishing Ltd, 2018.

[101] https://www.electronjs.org/docs/tutorial/about, 2020.

[102] https://keras.io/, 2019.

[103] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: A system for large-scale machine learning," in *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, 2016, pp. 265–283.

[104] J. Bergstra, F. Bastien, O. Breuleux, P. Lamblin, R. Pascanu, O. Delalleau, G. Desjardins, D. Warde-Farley, I. Goodfellow, A. Bergeron *et al.*, "Theano: Deep learning on gpus with python," in *NIPS 2011, BigLearning Workshop, Granada, Spain*, vol. 3. Citeseer, 2011, pp. 1–48.

[105] J. L. Rebelo Moreira, L. Ferreira Pires, and M. Van Sinderen, "Semantic interoperability for the iot: Analysis of json for linked data," *Enterprise Interoperability: Smart Services and Business Impact of Enterprise Interoperability*, pp. 163–169, 2018.

[106] K. Douzis, S. Sotiriadis, E. G. Petrakis, and C. Amza, "Modular and generic iot management on the cloud," *Future Generation Computer Systems*, vol. 78, pp. 369–378, 2018.

[107] P. Agarwal and M. Alam, "Investigating iot middleware platforms for smart application development," *arXiv preprint arXiv:1810.12292*, 2018.

[108] P. P. Ray, "Internet of robotic things: Concept, technologies, and challenges," *IEEE Access*, vol. 4, pp. 9489–9500, 2016.

[109] https://netmarketshare.com/operating-system-market-share.aspx?id=platformsDesktopVersions, 2019.

[110] H. Shiizuka and A. Hashizume, "The role of kansei/affective engineering and its expected in aging society," in *Intelligent Decision Technologies*. Springer, 2011, pp. 329–339.

[111] A. Jaimes, N. Sebe, and D. Gatica-Perez, "Human-centered computing: a multimedia perspective," in *Proceedings of the 14th ACM international conference on Multimedia*. ACM, 2006, pp. 855–864.

[112] L. Bannon, "Reimagining hci: toward a more human-centered perspective," *interactions*, vol. 18, no. 4, pp. 50–57, 2011.

[113] J. Kuffner, "Cloud-enabled robots in: Ieee-ras international conference on humanoid robots," *Piscataway, NJ: IEEE*, 2010.

[114] B. Kehoe, S. Patil, P. Abbeel, and K. Goldberg, "A survey of research on cloud robotics and automation," *IEEE Transactions on automation science and engineering*, vol. 12, no. 2, pp. 398–409, 2015.

[115] G. Mohanarajah, D. Hunziker, R. D'Andrea, and M. Waibel, "Rapyuta: A cloud robotics platform," *IEEE Transactions on Automation Science and Engineering*, vol. 12, no. 2, pp. 481–493, 2014.

[116] M. Beetz, M. Tenorth, and J. Winkler, "Open-ease," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 1983–1990.

[117] S. P. Pattar, E. Coronado, L. R. Ardila, and G. Venture, "Intention and engagement recognition for personalized human-robot interaction, an integrated and deep learning approach," in *2019 IEEE 4th International Conference on Advanced Robotics and Mechatronics (ICARM)*. IEEE, 2019, pp. 93–98.

[118] P. Fankhauser, M. Bloesch, D. Rodriguez, R. Kaestner, M. Hutter, and R. Siegwart, "Kinect v2 for mobile robot navigation: Evaluation and modeling," in *2015 International Conference on Advanced Robotics (ICAR)*. IEEE, 2015, pp. 388–394.

[119] V. H. Andaluz, W. X. Quevedo, F. A. Chicaiza, J. Varela, C. Gallardo, J. S. Sánchez, and O. Arteaga, "Transparency of a bilateral tele-operation scheme of a mobile manipulator robot," in *International Conference on Augmented Reality, Virtual Reality and Computer Graphics*. Springer, 2016, pp. 228–245.

[120] C. Bartneck, M. Soucy, K. Fleuret, and E. B. Sandoval, "The robot engine—making the unity 3d game engine work for hri," in *2015 24th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*. IEEE, 2015, pp. 431–437.

[121] I. Pakrasi, N. Chakraborty, and A. LaViers, "A design methodology for abstracting character archetypes onto robotic systems," in *Proceedings of the 5th International Conference on Movement and Computing*. ACM, 2018, p. 24.

[122] M. E. Segura, E. Coronado, M. Maya, A. Cardenas, and D. Piovesan, "Analysis of recoverable falls via microsoft kinect: Identification of third-order ankle dynamics," *Journal of Dynamic Systems, Measurement, and Control*, vol. 138, no. 9, p. 091006, 2016.

[123] https://www.newtonsoft.com/json, 2019.

[124] M. T. Jones, *BSD sockets programming from a multi-language perspective*. Charles River Media, Inc., 2003.

[125] S. Sechrest, "An introductory 4.4 bsd interprocess communication tutorial," *Computer Science Research Group, Department of Electrical Engineering and Computer Science, University of California, Berkeley*, 1986.

[126] E. Coronado, F. Mastrogiovanni, and G. Venture, "Development of intelligent behaviors for social robots via user-friendly and modular programming tools," in *Advanced Robotics and its Social Impacts (ARSO), 2018 EEE International Workshop on*. IEEE, 2018.

[127] P. Corke, "Integrating ros and matlab [ros topics]," *IEEE Robotics & Automation Magazine*, vol. 22, no. 2, pp. 18–20, 2015.

[128] Y. Hold-Geoffroy, M.-A. Gardner, C. Gagné, M. Latulippe, and P. Giguere, "ros4mat: A matlab programming interface for remote operations of ros-based robotic devices in an educational context," in *2013 International Conference on Computer and Robot Vision*. IEEE, 2013, pp. 242–248.

[129] https://enriquecoronadozu.github.io/NEP/, 2020.

[130] Y. Maruyama, S. Kato, and T. Azumi, "Exploring the performance of ros2," in *Proceedings of the 13th International Conference on Embedded Software*, 2016, pp. 1–10.

[131] H. Friedrich, S. Münch, R. Dillmann, S. Bocionek, and M. Sassin, "Robot programming by demonstration (rpd): Supporting the induction by human interaction," *Machine Learning*, vol. 23, no. 2, pp. 163–189, May 1996.

[132] S. Calinon, F. Guenter, and A. Billard, "On learning, representing, and generalizing a task in a humanoid robot," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 37, no. 2, pp. 286–298, April 2007.

[133] J. F. Gorostiza and M. A. Salichs, "Natural programming of a social robot by dialogs." in *AAAI Fall Symposium: Dialog with Robots*, 2010.

[134] B. R. Barricelli and S. Valtolina, "A visual language and interactive system for end-user development of internet of things ecosystems," *Journal of Visual Languages & Computing*, vol. 40, pp. 1–19, 2017.

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

[136] I. Sagredo-Olivenza, P. P. Gómez-Martín, M. A. Gómez-Martín, and P. A. González-Calero, "Combining neural networks for controlling non-player characters in games," in *International Work-Conference on Artificial Neural Networks*. Springer, 2017, pp. 694–705.

[137] R. Francese, M. Risi, and G. Tortora, "Iconic languages: Towards end-user programming of mobile applications," *Journal of Visual Languages & Computing*, vol. 38, pp. 1–8, 2017.

[138] J. M. Mota, I. Ruiz-Rube, J. M. Dodero, and I. Arnedillo-Sánchez, "Augmented reality mobile app development for all," *Computers & Electrical Engineering*, vol. 65, pp. 250–260, 2018.

[139] D. Budgen and P. Brereton, "Performing systematic literature reviews in software engineering," in *Proceedings of the 28th international conference on Software engineering*. ACM, 2006, pp. 1051–1052.

[140] B. Kitchenham, "Procedures for performing systematic reviews," vol. 33, pp. 1–26, 2004.

[141] D. Tetteroo and P. Markopoulos, "A review of research methods in end user development," in *International Symposium on End User Development*. Springer, 2015, pp. 58–75.

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

[143] F. Paternò, "End user development: Survey of an emerging field for empowering people," *ISRN Software Engineering*, vol. 2013, 2013.

[144] M. Santos and M. L. B. Villela, "Characterizing end-user development solutions: A systematic literature review," in *International Conference on Human-Computer Interaction*. Springer, 2019, pp. 194–209.

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

[146] A. Bellucci, A. Vianello, Y. Florack, L. Micallef, and G. Jacucci, "Augmenting objects at home through programmable sensor tokens: A design journey," *International Journal of Human-Computer Studies*, vol. 122, pp. 211 – 231, 2019.

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

[148] M. Colledanchise and P. Ögren, *Behavior Trees in Robotics and Al: An Introduction*. CRC Press, 2018.

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

[150] F. A. Bravo, A. M. González, and E. González, "A review of intuitive robot programming environments for educational purposes," in *2017 IEEE 3rd Colombian Conference on Automatic Control (CCAC)*, Oct 2017, pp. 1–6.

[151] M. E. Karim, S. Lemaignan, and F. Mondada, "A review: Can robots reshape k-12 stem education?" in *2015 IEEE International Workshop on Advanced Robotics and its Social Impacts (ARSO)*, June 2015, pp. 1–8.

[152] G. Golovchinsky, "Cognitive dimensions analysis of interfaces for information seeking," *arXiv preprint arXiv:0908.3523*, 2009.

[153] T. Green and M. Petre, "Usability analysis of visual programming environments: A 'cognitive dimensions' framework," *Journal of Visual Languages & Computing*, vol. 7, no. 2, pp. 131 – 174, 1996.

[154] J. Dagit, J. Lawrance, C. Neumann, M. Burnett, R. Metoyer, and S. Adams, "Using cognitive dimensions: advice from the trenches," *Journal of Visual Languages & Computing*, vol. 17, no. 4, pp. 302–327, 2006.

[155] T. Green and A. Blackwell, "Cognitive dimensions of information artefacts: a tutorial," in *BCS HCI Conference*, vol. 98, 1998.

[156] J. Diprose, B. MacDonald, J. Hosking, and B. Plimmer, "Designing an API at an appropriate abstraction level for programming social robot applications," *Journal of Visual Languages & Computing*, vol. 39, pp. 22–40, 2017.

[157] A. Carfi, J. Villalobos, E. Coronado, B. Bruno, and F. Mastrogiovanni, "Can human-inspired learning behaviour facilitate human–robot interaction?" *International Journal of Social Robotics*, vol. 1, pp. 1 – 14, 2019.

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

[159] D. Weintrop, D. C. Shepherd, P. Francis, and D. Franklin, "Blockly goes to work: Block-based programming for industrial robots," in *2017 IEEE Blocks and Beyond Workshop (B B)*, Oct 2017, pp. 29–36.

[160] F. Steinmetz, A. Wollschläger, and R. Weitschat, "Razer–a human-robot interface for visual task-level programming and intuitive skill parametrization," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 1362–1369, 2018.

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

[162] G. Serafini, "Teaching programming at primary schools: Visions, experiences, and long-term research prospects," in *Informatics in Schools. Contributing to 21st Century Education*, I. Kalaš and R. T. Mittermeir, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 143–154.

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

[164] D. S. Touretzky and C. Gardner-McCune, "Calypso for cozmo: Robotic ai for everyone (abstract only)," in *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, 2018, pp. 1110–1110.

[165] J. Shin, R. Siegwart, and S. Magnenat, "Visual programming language for thymio ii robot," in *Conference on Interaction Design and Children (IDC'14)*. ETH Zürich, 2014.

[166] M. F. Costabile, D. Fogli, G. Fresta, P. Mussio, and A. Piccinno, "Software environments for end-user development and tailoring." *PsychNology Journal*, vol. 2, no. 1, pp. 99–122, 2004.

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

[168] T. Fong, I. Nourbakhsh, and K. Dautenhahn, "A survey of socially interactive robots," *Robotics and Autonomous Systems*, vol. 42, no. 3, pp. 143 – 166, 2003, socially Interactive Robots.

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

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

[171] N. Leonardi, M. Manca, F. Paternò, and C. Santoro, "Trigger-action programming for personalising humanoid robot behaviour," in *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. ACM, 2019, p. 445.

[172] F. Corno, L. De Russis, and A. Monge Roffarello, "Empowering end users in debugging trigger-action rules," in *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. ACM, 2019, p. 388.

[173] S. Gaudl, "Building robust real-time game ai: simplifying & automating integral process steps in multi-platform design." Ph.D. dissertation, University of Bath, 2016.

[174] M. Colledanchise and P. Ögren, "How behavior trees modularize hybrid control systems and generalize sequential behavior compositions, the subsumption architecture, and decision trees," *IEEE Transactions on Robotics*, vol. 33, no. 2, pp. 372–389, April 2017.

[175] J. Bohren and S. Cousins, "The smach high-level executive [ros news]," *IEEE Robotics & Automation Magazine*, vol. 17, no. 4, pp. 18–20, Dec 2010.

[176] D. D. Hils, "Visual languages and computing survey: Data flow visual programming languages," *Journal of Visual Languages & Computing*, vol. 3, no. 1, pp. 69 – 101, 1992.

[177] D. Weintrop and U. Wilensky, "Comparing block-based and text-based programming in high school computer science classrooms," *ACM Transactions on Computing Education (TOCE)*, vol. 18, no. 1, pp. 1–25, 2017.

[178] D. Weintrop, "Block-based programming in computer science education," *Communications of the ACM*, vol. 62, no. 8, pp. 22–25, 2019.

[179] M. Resnick, J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. Silver, B. Silverman *et al.*, "Scratch: programming for all," *Communications of the ACM*, vol. 52, no. 11, pp. 60–67, Nov. 2009.

[180] D. Garcia, L. Segars, and J. Paley, "Snap!(build your own blocks): tutorial presentation," *Journal of Computing Sciences in Colleges*, vol. 27, no. 4, pp. 120–121, 2012.

[181] https://developers.google.com/blockly/, 2018.

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

[183] T. Schulz, J. Torresen, and J. Herstad, "Animation techniques in human-robot interaction user studies: A systematic literature review," *ACM Transactions on Human-Robot Interaction (THRI)*, vol. 8, no. 2, pp. 1–22, 2019.

[184] A. R. Martin and S. Colton, "Towards liveness in game development," in *2019 IEEE Conference on Games (CoG)*. IEEE, 2019, pp. 1–4.

[185] P. Rein, S. Ramson, J. Lincke, R. Hirschfeld, and T. Pape, "Exploratory and live, programming and coding: A literature study comparing perspectives on liveness," *arXiv preprint arXiv:1807.08578*, 2018.

[186] M. Campusano and J. Fabry, "Live robot programming: The language, its implementation, and robot api independence," *Science of Computer Programming*, vol. 133, pp. 1–19, 2017.

[187] S. L. Tanimoto, "Viva: A visual language for image processing," *Journal of Visual Languages & Computing*, vol. 1, no. 2, pp. 127–139, 1990.

[188] L. D. Riek, "Wizard of oz studies in hri: a systematic review and new reporting guidelines," *Journal of Human-Robot Interaction*, vol. 1, no. 1, pp. 119–136, 2012.

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

[190] E. Pot, J. Monceaux, R. Gelin, and B. Maisonnier, "Choregraphe: a graphical tool for humanoid robot programming," in *RO-MAN 2009 - The 18th IEEE International Symposium on Robot and Human Interactive Communication*, Sep. 2009, pp. 46–51.

[191] http://tools.seobook.com/general/keyword-density/, 2019.

[192] H. Zhang, M. A. Babar, and P. Tell, "Identifying relevant studies in software engineering," *Information and Software Technology*, vol. 53, no. 6, pp. 625–637, 2011.

[193] 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.

[194] S. Keele *et al.*, "Guidelines for performing systematic literature reviews in software engineering," Technical report, Ver. 2.3 EBSE Technical Report. EBSE, Tech. Rep., 2007.

[195] K. Petersen, S. Vakkalanka, and L. Kuzniarz, "Guidelines for conducting systematic mapping studies in software engineering: An update," *Information and Software Technology*, vol. 64, pp. 1–18, 2015.

[196] P. Brereton, B. A. Kitchenham, D. Budgen, M. Turner, and M. Khalil, "Lessons from applying the systematic literature review process within the software engineering domain," *Journal of systems and software*, vol. 80, no. 4, pp. 571–583, 2007.

[197] J. Jackson, "Microsoft robotics studio: A technical introduction," *IEEE Robotics Automation Magazine*, vol. 14, no. 4, pp. 82–87, Dec 2007.

[198] https://github.com/hcrlab/code_it, 2018.

[199] B. Jost, M. Ketterl, R. Budde, and T. Leimbach, "Graphical programming environments for educational robots: Open roberta - yet another one?" in *2014 IEEE International Symposium on Multimedia*, Dec 2014, pp. 381–386.

[200] I. Zubrycki and G. Granosik, "Designing an interactive device for sensory therapy," in *2016 11th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, March 2016, pp. 545–546.

[201] M. Seraj, S. Autexier, and J. Janssen, "Beesm, a block-based educational programming tool for end users," in *Proceedings of the 10th Nordic Conference on Human-Computer Interaction*, ser. NordiCHI '18. ACM, 2018, pp. 886–891.

[202] https://enriquecoronadozu.github.io/RIZE/, 2018.

[203] P. Ziafati, F. Lera, A. Costa, A. Nazarikhorram, L. Van Der Torre, and A. Nazarikhor, "Procrob architecture for personalized social robotics," in *Robots for Learning Workshop@ HRI*, 2017, pp. 6–9.

[204] C. Datta and B. A. MacDonald, "Architecture of an extensible visual programming environment for authoring behaviour of personal service robots," in *2017 First IEEE International Conference on Robotic Computing (IRC)*, April 2017, pp. 156–159.

[205] F. Erich, M. Hirokawa, and K. Suzuki, "A visual environment for reactive robot programming of macro-level behaviors," in *Social Robotics*. Springer, 2017, pp. 577–586.

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

[207] A. Sauppé and B. Mutlu, "Design patterns for exploring and prototyping human-robot interactions," in *Proceedings of the 32Nd Annual ACM Conference on Human Factors in Computing Systems*, ser. CHI '14. ACM, 2014, pp. 1439–1448.

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

[209] http://https://www.qt.io/, 2018.

[210] M. H. Lee, H. S. Ahn, K. Wang, and B. MacDonald, "Design of an API for integrating robotic software frameworks," in *Proceedings of the 2014 Australasian Conference on Robotics and Automation (ACRA 2014)*, vol. 2, no. 3.2, 2014, p. 1.

[211] H. Bruyninckx, "Open robot control software: the orocos project," in *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)*, vol. 3, May 2001, pp. 2523–2528 vol.3.

[212] C. Datta, H. Y. Yang, I. Kuo, E. Broadbent, and B. A. MacDonald, "Software platform design for personal service robots in healthcare," in *2013 6th IEEE Conference on Robotics, Automation and Mechatronics (RAM)*, Nov 2013, pp. 156–161.

[213] S. Tilkov and S. Vinoski, "Node.js: Using javascript to build high-performance network programs," *IEEE Internet Computing*, vol. 14, no. 6, pp. 80–83, Nov 2010.

[214] E. Bainomugisha, A. L. Carreton, T. v. Cutsem, S. Mostinckx, and W. d. Meuter, "A survey on reactive programming," *ACM Computing Surveys (CSUR)*, vol. 45, no. 4, p. 52, Aug. 2013.

[215] https://github.com/FlorisE/RRP, 2018.

[216] M. Kwiatkowska, G. Norman, and D. Parker, "Prism 4.0: Verification of probabilistic real-time systems," in *Computer Aided Verification*. Berlin, Heidelberg: Springer, 2011, pp. 585–591.

[217] https://github.com/Wisc-HCI/RoVer, 2018.

[218] A. Haddadi and K. Sundermeyer, "Foundations of distributed artificial intelligence," G. M. P. O'Hare and N. R. Jennings, Eds. John Wiley & Sons, Inc., 1996, ch. Belief-desire-intention Agent Architectures, pp. 169–185.

[219] J. Huang, T. Lau, and M. Cakmak, "Design and evaluation of a rapid programming system for service robots," in *2016 11th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, March 2016, pp. 295–302.

[220] V. Paramasivam, J. Huang, S. Elliott, and M. Cakmak, "Computer science outreach with end-user robot-programming tools," in *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, ser. SIGCSE '17. ACM, 2017, pp. 447–452.

[221] B. Wulff, A. Wilson, B. Jost, and M. Ketterl, "An adopter centric api and visual programming interface for the definition of strategies for automated camera tracking," in *2015 IEEE International Symposium on Multimedia (ISM)*, Dec 2015, pp. 587–592.

[222] https://snap.berkeley.edu/, 2018.

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

[224] L. Rincon, E. Coronado, H. Hendra, J. Phan, Z. Zainalkefli, and G. Venture, "Expressive states with a robot arm using adaptive fuzzy and robust predictive controllers," in *2018 3rd International Conference on Control and Robotics Engineering (ICCRE)*, April 2018, pp. 11–15.

[225] J. Forcier, P. Bissex, and W. J. Chun, *Python web development with Django*. Addison-Wesley Professional, 2008.

[226] S. Vinoski, "Server-sent events with yaws," *IEEE internet computing*, vol. 16, no. 5, pp. 98–102, 2012.

[227] T. J. Bench-Capon, *Knowledge representation: An approach to artificial intelligence*. Elsevier, 2014, vol. 32.

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

[229] M. Dawe, S. Garolinski, L. Dicken, T. Humphreys, and D. Mark, "Behavior selection algorithms: an overview," *Game AI Pro: Collected wisdom of game ai professionals. CRC Press, S*, pp. 47–60, 2014.

[230] E. W. Dijkstra, "Letters to the editor: go to statement considered harmful," *Communications of the ACM*, vol. 11, no. 3, pp. 147–148, 1968.

[231] K. Finstad, "The system usability scale and non-native english speakers," *Journal of usability studies*, vol. 1, no. 4, pp. 185–188, 2006.

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

[233] R. Rajkumar, M. Gagliardi, and Lui Sha, "The real-time publisher/subscriber interprocess communication model for distributed real-time systems: design and implementation," in *Proceedings Real-Time Technology and Applications Symposium*, May 1995, pp. 66–75.

[234] A. S. Tanenbaum and M. Van Steen, *Distributed systems: principles and paradigms*. Prentice-Hall, 2007.

[235] A. Orebäck and H. I. Christensen, "Evaluation of architectures for mobile robotics," *Autonomous Robots*, vol. 14, no. 1, pp. 33–49, Jan 2003. [Online]. Available: doi.org/10.1023/A:1020975419546

[236] W. Gellerich, M. Kosiol, and E. Ploedereder, "Where does goto go to?" in *International Conference on Reliable Software Technologies*. Springer, 1996, pp. 385–395.

[237] P. Janssen, "Visual dataflow modelling-some thoughts on complexity," 2014.

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

[239] I. Leite, C. Martinho, and A. Paiva, "Social robots for long-term interaction: A survey," *International Journal of Social Robotics*, vol. 5, no. 2, pp. 291–308, Apr 2013.

[240] S. Wang and H. I. Christensen, "Tritonbot: First lessons learned from deployment of a long-term autonomy tour guide robot," in *2018 27th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, Aug 2018, pp. 158–165.

[241] A. F. Blackwell, C. Britton, A. Cox, T. R. G. Green, C. Gurr, G. Kadoda, M. S. Kutar, M. Loomes, C. L. Nehaniv, M. Petre, C. Roast, C. Roe, A. Wong, and R. M. Young, "Cognitive dimensions of notations: Design tools for cognitive technology," in *Cognitive Technology: Instruments of Mind*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 325–341.

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

[243] A. Assila, H. Ezzedine *et al.*, "Standardized usability questionnaires: Features and quality focus," *Electronic Journal of Computer Science and Information Technology: eJCIST*, vol. 6, no. 1, 2016.

[244] J. Brooke *et al.*, "Sus-a quick and dirty usability scale," *Usability evaluation in industry*, vol. 189, no. 194, pp. 4–7, 1996.

[245] "Development of nasa-tlx (task load index): Results of empirical and theoretical research," in *Human Mental Workload*, ser. Advances in Psychology, P. A. Hancock and N. Meshkati, Eds. North-Holland, 1988, vol. 52, pp. 139 – 183.

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

[247] https://www.usability.gov/what-and-why/user-experience.html, 2018.

[248] F. Mastrogiovanni, A. Sgorbissa, and R. Zaccaria, "A system for hierarchical planning in service mobile robotics," in *Proceedings of the Eight Conference on Intelligent Autonomous Systems (IAS-8)*, Amsterdam, The Netherlands, March 2004.

[249] A. Capitanelli, M. Maratea, F. Mastrogiovanni, and M. Vallati, "On the manipulation of articulated objects in human–robot cooperation scenarios," *Robotics and Autonomous Systems*, vol. 109, pp. 139 – 155, 2018.

[250] D. A. Norman, "Design principles for human-computer interfaces," in *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*. ACM, 1983, pp. 1–10.

[251] J. Johnson, *Designing with the mind in mind: simple guide to understanding user interface design guidelines*. Elsevier, 2013.

[252] A. Altaboli and Y. Lin, "Objective and subjective measures of visual aesthetics of website interface design: the two sides of the coin," *Human-computer interaction. Design and development approaches*, pp. 35–44, 2011.

[253] D. A. Norman, *Emotional design: Why we love (or hate) everyday things*. Basic Civitas Books, 2004.

[254] https://vuetifyjs.com/, 2020.

[255] https://material.io/design/, 2020.

[256] P. H. Kahn, N. G. Freier, T. Kanda, H. Ishiguro, J. H. Ruckert, R. L. Severson, and S. K. Kane, "Design patterns for sociality in human-robot interaction," in *Proceedings of the 3rd ACM/IEEE international conference on Human robot interaction*. ACM, 2008, pp. 97–104.

[257] E. Cha, J. Forlizzi, and S. S. Srinivasa, "Robots in the home: Qualitative and quantitative insights into kitchen organization," in *Proceedings of the Tenth Annual ACM/IEEE International Conference on Human-Robot Interaction*. ACM, 2015, pp. 319–326.

[258] P. Uluer, N. Akalın, and H. Köse, "A new robotic platform for sign language tutoring," *International Journal of Social Robotics*, vol. 7, no. 5, pp. 571–585, 2015.

[259] E. Jochum, E. Vlachos, A. Christoffersen, S. G. Nielsen, I. A. Hameed, and Z.-H. Tan, "Using theatre to study interaction with care robots," *International Journal of Social Robotics*, vol. 8, no. 4, pp. 457–470, 2016.

[260] F. Jumel, J. Saraydaryan, R. Leber, L. Matignon, E. Lombardi, C. Wolf, and O. Simonin, "Context aware robot architecture, application to the robocup@ home challenge," in *RoboCup symposium*, 2018.

[261] M. de Jong, K. Zhang, A. M. Roth, T. Rhodes, R. Schmucker, C. Zhou, S. Ferreira, J. Cartucho, and M. Veloso, "Towards a robust interactive and learning social robot," in *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, 2018, pp. 883–891.

[262] B. Alenljung, J. Lindblom, R. Andreasson, and T. Ziemke, "User experience in social human-robot interaction," *International Journal of Ambient Computing and Intelligence (IJACI)*, vol. 8, no. 2, pp. 12–31, 2017.

[263] M. M. de Graaf, S. Ben Allouch, and J. A. van Dijk, "A phased framework for long-term user acceptance of interactive technology in domestic environments," *New Media & Society*, p. 1461444817727264, 2017.

[264] M. Dziergwa, M. Kaczmarek, P. Kaczmarek, J. Kkedzierski, and K. Wadas-Szydłowska, "Long-term cohabitation with a social robot: a case study of the influence of human attachment patterns," *International Journal of Social Robotics*, vol. 10, no. 1, pp. 163–176, 2018.

[265] M. M. de Graaf, S. B. Allouch, and J. A. van Dijk, "Long-term evaluation of a social robot in real homes," *Interaction studies*, vol. 17, no. 3, pp. 462–491, 2017.

[266] Y. Fernaeus, M. Håkansson, M. Jacobsson, and S. Ljungblad, "How do you play with a robotic toy animal?: a long-term study of pleo," in *Proceedings of the 9th international Conference on interaction Design and Children*. ACM, 2010, pp. 39–48.

[267] J. Fink, V. Bauwens, F. Kaplan, and P. Dillenbourg, "Living with a vacuum cleaning robot," *International Journal of Social Robotics*, vol. 5, no. 3, pp. 389–408, Aug 2013. [Online]. Available: https://doi.org/10.1007/s12369-013-0190-2

[268] S. Frennert, H. Eftring, and B. Östlund, "Case report: Implications of doing research on socially assistive robots in real homes," *International Journal of Social Robotics*, vol. 9, no. 3, pp. 401–415, 2017.

[269] M. Vincze, M. Bajones, M. Suchi, D. Wolf, L. Lammer, A. Weiss, and D. Fischinger, "User experience results of setting free a service robot for older adults at home," in *Service Robots*. InTech, 2018.

[270] M. Sustrik, "Distributed Computing: The Survey Pattern," http://250bpm.com/blog:5, 2018, [Online; accessed 01-06-2018].

[271] A. A. Ramirez-Duque, A. Frizera-Neto, and T. F. Bastes, "Robot-assisted diagnosis for children with autism spectrum disorder based on automated analysis of nonverbal cues," in *2018 7th IEEE International Conference on Biomedical Robotics and Biomechatronics (Biorob)*. IEEE, 2018, pp. 456–461.

[272] M. Alemi, A. Ghanbarzadeh, A. Meghdari, and L. J. Moghadam, "Clinical application of a humanoid robot in pediatric cancer interventions," *International Journal of Social Robotics*, vol. 8, no. 5, pp. 743–759, 2016.

[273] D. O. David, C. A. Costescu, S. Matu, A. Szentagotai, and A. Dobrean, "Developing joint attention for children with autism in robot-enhanced therapy," *International Journal of Social Robotics*, pp. 1–11, 2018.

[274] H. Peng, J. Li, H. Hu, C. Zhou, and Y. Ding, "Robotic choreography inspired by the method of human dance creation," *Information*, vol. 9, no. 10, p. 250, 2018.

[275] I. Infantino, A. Augello, A. Manfré, G. Pilato, and F. Vella, "Robodanza: Live performances of a creative dancing humanoid," in *Proceedings of the Seventh International Conference on Computational Creativity*, 2016.

[276] R. Ros, M. Nalin, R. Wood, P. Baxter, R. Looije, Y. Demiris, T. Belpaeme, A. Giusti, and C. Pozzi, "Child-robot interaction in the wild: advice to the aspiring experimenter," in *Proceedings of the 13th international conference on multimodal interfaces*. ACM, 2011, pp. 335–342.

[277] J. Zhang, J. Zheng, and N. Magnenat-Thalmann, "Modeling personality, mood, and emotions," in *Context Aware Human-Robot and Human-Agent Interaction*. Springer, 2016, pp. 211–236.

[278] J. A. Russell and A. Mehrabian, "Evidence for a three-factor theory of emotions," *Journal of research in Personality*, vol. 11, no. 3, pp. 273–294, 1977.

[279] M. Karg, A.-A. Samadani, R. Gorbet, K. Kühnlenz, J. Hoey, and D. Kulić, "Body movements for affective expression: A survey of automatic recognition and generation," *IEEE Transactions on Affective Computing*, vol. 4, no. 4, pp. 341–359, 2013.

[280] M. M. Bradley and P. J. Lang, "Measuring emotion: the self-assessment manikin and the semantic differential," *J. of behavior therapy and experimental psychiatry*, vol. 25, no. 1, pp. 49–59, 1994.

[281] A. Van Der Heide, D. Sánchez, and G. Trivino, "Computational models of affect and fuzzy logic," in *Proceedings of the 7th Conference of the European Society for Fuzzy Logic and Technology*. Atlantis Press, 2011, pp. 620–627.

[282] C. J. Willemse, D. K. Heylen, and J. B. van Erp, "Warmth in affective mediated interaction: Exploring the effects of physical warmth on interpersonal warmth," in *Affective Computing and Intelligent Interaction (ACII), 2015 Int. Conf. on*. IEEE, 2015, pp. 28–34.

[283] J. C. Baxter, "Interpersonal spacing in natural settings," *Sociometry*, pp. 444–456, 1970.

[284] N. S. Altman, "An introduction to kernel and nearest-neighbor nonparametric regression," *The American Statistician*, vol. 46, no. 3, pp. 175–185, 1992.

[285] https://biz.nikkan.co.jp/eve/irex/english/, 2019.

[286] A. Hashizume and M. Kurosu, "Kansei engineering as an indigenous research field originated in japan," in *International Conference on Human-Computer Interaction*. Springer, 2016, pp. 46–52.

[287] M. Nagamachi, *Kansei/affective engineering*. crc press, 2016.

[288] ——, "Kansei engineering: a new ergonomic consumer-oriented technology for product development," *International Journal of industrial ergonomics*, vol. 15, no. 1, pp. 3–11, 1995.

[289] M. Nagamachi and A. M. Lokman, *Innovations of Kansei engineering*. CRC Press, 2016.

[290] Z. Yahya and M. Y. Shafazand, "Kansei design customization based on personality modelling," in *International Conference on Kansei Engineering & Emotion Research*. Springer, 2018, pp. 399–407.

[291] N. Mitsuo, "Perspectives and new trend of kansei/affective engineering," in *1st European Conference on Affective Design and Kansei Engineering & 10th QMOD Conference, University of Linkoping and Lund University, Helsingborg*, 2007.

[292] M. Nagamachi, M. Tachikawa, N. Imanishi, T. Ishizawa, and S. Yano, "A successful statistical procedure on kansei engineering products," in *11th QMOD Conference. Quality Management and Organizational Development Attaining Sustainability From Organizational Excellence to SustainAble Excellence; 20-22 August; 2008 in Helsingborg; Sweden*, no. 033. Linköping University Electronic Press, 2008, pp. 987–995.

[293] A. M. Lokman, M. B. C. Haron, S. Z. Z. Abidin, N. E. A. Khalid, and S. Ishihara, "Prelude to natphoric kansei engineering framework," *Journal of Software Engineering and Applications*, vol. 6, no. 12, p. 638, 2013.

[294] J. Al-Hindawe *et al.*, "Considerations when constructing a semantic differential scale," *La Trobe papers in linguistics*, vol. 9, no. 7, pp. 1–9, 1996.

[295] J.-A. Claret, G. Venture, and L. Basañez, "Exploiting the robot kinematic redundancy for emotion conveyance to humans as a lower priority task," *Int. J. of social robotics*, vol. 9, no. 2, pp. 277–292, 2017.