# Can ChatGPT Support End-User Development of Robot Programs?

Giorgio Bimbatti[1,†], Daniela Fogli[1,†] and Luigi Gargioni[1,*,†]

[1]*Department of Information Engineering, University of Brescia, Via Branze 38, Brescia, Italy*

Abstract
In this paper, we investigate the use of OpenAI ChatGPT to improve the natural language understanding of an End-User Development environment, called CAPIRCI, supporting users neither expert in computer programming nor expert in robotics to create programs for a collaborative robot. The integration of ChatGPT in CAPIRCI is studied to be transparent for the user, who will be allowed to check and correct that program description generated by ChatGPT, by interacting with an intuitive block-based interface, according to a Human-Centered Artificial Intelligence design approach.

**Keywords**
Collaborative robot, Robot programming, AI-based system, Human-centered AI

## 1. Introduction

Collaborative robots (also known as *cobots*) are recently being deployed to automate tasks in several domains — from office work to manufacturing, from logistics to healthcare. They can operate in the same space of human workers and collaborate with them to achieve shared goals. Collaborative robots contribute to create the conditions for increasing production flexibility, being more affordable, compact and easy-to-use than traditional industrial robots [1]. However, notwithstanding their potential, barriers exist in their wide adoption due to the complexity of programming their tasks. Producers of cobots are thus implementing physical interaction techniques with robots and programming tools aimed at supporting easy definition of simple robot tasks by users without any experience of robotics and software programming. However, these techniques and tools are still far away from the ideal solution, and several research scholars are studying new approaches to solving this problem, even though these approaches often require that users have some technical background (e.g., [2, 3]), or are evaluated with computer science/engineering students (e.g., [4]), rather than with end users having limited computational fluency [5].

In [6, 7], we proposed CAPIRCI (Chat And Program Industrial Robots through Convenient Interaction), an End-User Development (EUD) [8, 9, 10] environment providing an intuitive and

*IS-EUD 2023: 9th International Symposium on End-User Development, 6-8 June 2023, Cagliari, Italy*
[*]Corresponding author.
[†]These authors contributed equally.

g.bimbatti@studenti.unibs.it (G. Bimbatti); daniela.fogli@unibs.it (D. Fogli); luigi.gargioni@unibs.it (L. Gargioni)
https://daniela-fogli.unibs.it (D. Fogli)
0000-0003-1479-2240 (D. Fogli)

natural way to program pick-and-place tasks for a collaborative robot. Specifically, CAPIRCI encompasses a hybrid interaction style that merges two paradigms: natural language interaction with a chat-based interface, and visual interaction with a block-based graphic interface.

The final aim of CAPIRCI is not simply demonstrating that the proposed interaction style for EUD of robot tasks is feasible and easier than block-based programming alone, as shown in the experiment carried out in [7], but also promoting a way to foster gradual learning of robot programming on behalf of human workers, thus facilitating a smooth acquisition of computational fluency.

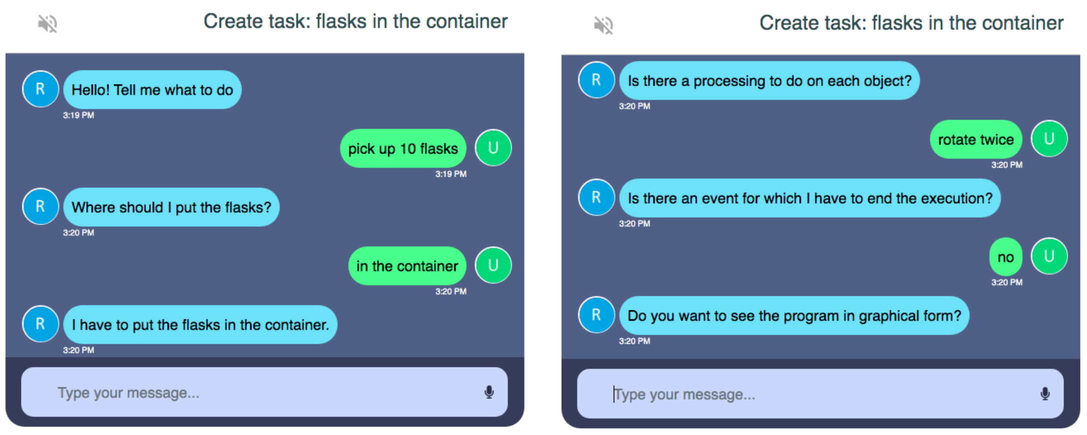
However, one of the main weaknesses of the CAPIRCI prototype presented in [7] was the computational power of the chat-based interface. Natural language processing (NLP) was implemented exploiting Python libraries, like the Standard CoreNLP and the NLTK (Natural Language ToolKit) package, and was based on the recognition of a finite set of noun domain-dependent phrases. In this paper, we explore the use of OpenAI ChatGPT[1] to improve the natural language understanding of our chat-based interface and show how ChatGPT could be of help in the case of robot programming.

## 2. CAPIRCI: a EUD environment for robot programming

The EUD feature of CAPIRCI is composed of 1) a chat environment where the system and the user interact through a simple guided natural language dialogue that leads to the definition of tasks for a collaborative robot, and 2) a visual environment where tasks can be composed through direct manipulation of specific types of blocks.

A robot task may include the specification of the objects to pick up, the actions that must be performed on the objects, and the locations where the objects must be put. Pick-and-place tasks may include repetitions if more than one object must be manipulated or the action must be carried out on the object several times. Conditional termination of tasks may depend on specific events. For instance, considering the scenario of flask manipulation in an analytical laboratory, Figure 1 shows a user-system dialogue in the chat environment aimed at defining a task for picking up 10 flasks and put them in a container after that each flask has been rotated twice, and where no specific event determines the end of task execution. The interaction with the chat always ends with a question to the user asking whether they would like to see the program in the visual environment. If the user answers affirmatively, they will have the possibility to verify the program created by means of the natural language dialogue in the visual environment, and possibly modify or extend it through drag-and-drop of predefined controls and functional blocks. Figure 2 shows the main parts of the visual environment. We denoted with (A) the part of the graphic interface where the user can find the libraries of components useful for task programming (*Tasks* — since tasks previously developed can be saved and re-used —, *Controls*, *Events*, *Actions*, *Objects*, and *Locations*). The library *Controls* is currently selected in the figure, and the blocks corresponding to the available control statements are shown at the right of the libraries; the user can select and drag-and-drop one of them in the working area denoted with (B) in the figure. In this working area, there is the visualization of the task just created with the chat. The two programming modes may accommodate different users' attitudes: users

[1]https://openai.com/blog/chatgpt/


Create task: flasks in the container

R: Hello! Tell me what to do
3:19 PM

U: pick up 10 flasks
3:19 PM

R: Where should I put the flasks?
3:20 PM

U: in the container
3:20 PM

R: I have to put the flasks in the container.
3:20 PM

Type your message...

Create task: flasks in the container

R: Is there a processing to do on each object?
3:20 PM

U: rotate twice
3:20 PM

R: Is there an event for which I have to end the execution?
3:20 PM

U: no
3:20 PM

R: Do you want to see the program in graphical form?
3:20 PM

Type your message...

**Figure 1:** The chat-based interface: the user would like to define a task to put 10 flasks in the container after having rotate each one twice.

with advanced computational fluency may prefer interacting only with the visual environment, which supports the creation of complex tasks including nested loops and conditional statements; users with more limited computational fluency, instead, may create a first draft of the robot program with the chat, which appears more intuitive and easy to use, and then use the visual environment to refine and complete the draft program.

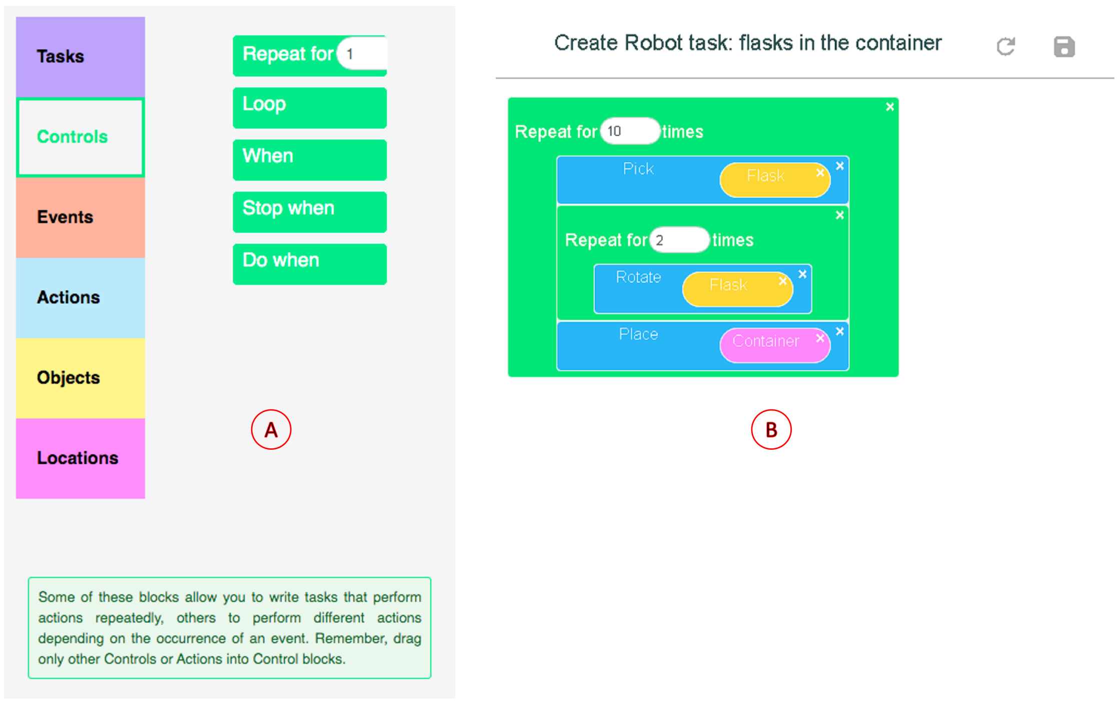
Each task created either with the chat environment or the visual environment is saved as an XML file in the *Tasks* library for future modifications or re-use in other robot tasks. The XML description of a task represents a formal specification of a robot program. To run the program on a real robot (e.g., the COBOTTA robot[2] by DENSO WAVE Ltd.), a parser analyzes the XML document to identify the elements (objects to be manipulated, actions to be executed, locations where to put the objects, loops to be performed, etc.) and generates the corresponding Python code. This code is interpreted at run time, along with a few proprietary libraries, to perform the task on the real robot.

## 3. Can ChatGPT Support End-User Development?

Analysing the features of CAPIRCI, one might argue that its chat environment could provide more powerful features, in order to allow all users to create complex tasks in a natural way, without resorting to the graphic interface. However, the visual representation of the program permits to check its correctness and completeness in an immediate way. Alternatively, the chat could generate directly the executable code corresponding to the robot task, but most of end users not expert in computer programming would be unable to understand and check it.

The idea of providing a natural language interface for robot programming has been investigated in literature for several years (e.g., [11, 12, 13]) and is currently gaining momentum thanks to the recent introduction of OpenAI ChatGPT, a pre-trained generative text model,

[2]https://www.densorobotics-europe.com/product-overview/products/colla borative-robots/cobotta



Some of these blocks allow you to write tasks that perform actions repeatedly, others to perform different actions depending on the occurrence of an event. Remember, drag only other Controls or Actions into Control blocks.

**Figure 2:** The graphic interface: the set of libraries is on the left (A); the task defined by the user to put 10 flasks in the container is on the right (B).

which provides impressive interaction capabilities. For instance, in [14], the authors investigate ChatGPT as a potential versatile tool for robot programming, by first creating a high-level function library and then allowing ChatGPT to parse user requests and convert them to a logical sequence of function calls. However, this work assigns to the user a significant and critical role underlining that "the user stays on the loop to evaluate code output by ChatGPT, either through direct analysis or through simulation, and provides feedback to ChatGPT on the quality and safety of the output code" [14]. According to this approach, after a few iterations of user-ChatGPT dialogue, the final code can be deployed on the robot. The assumption here is that the user is able to understand the generated code, to assess its correctness, and to suggest possible modifications, if needed.

In our approach, we would like to keep a natural language dialogue to generate a first draft of the program for the robot and provide the user with the possibility of visualizing this draft program in a graphic interface, so to modify it, if necessary, in an intuitive manner, that is, with drag-and-drop and direct operation on the graphic visualization of the draft program. This is even more important when created programs must be executed on a robot, as the safety of the human and the environment, as well as of the robot itself, is at stake.

To this end, our idea is integrating ChatGPT features in CAPIRCI, by substituting the NLP Python libraries exploited in the first prototype and using ChatGPT to generate the XML intermediate description of robot tasks.

## 4. Integrating CAPIRCI with ChatGPT

The basic idea for integrating the powerful NLP capabilities of ChatGPT in CAPIRCI was creating a new software layer able to acquire the user's request in natural language defining a specific task for a collaborative robot and formulate the correct question to ChatGPT to obtain the XML file describing the robot task. The XML file must be syntactically structured according to the rules specified in CAPIRCI for its correct interpretation both by the parser functionalities that generate the graphic representation of tasks and by those ones able to generate the Python code for run time execution.

Preliminary tests have been performed with ChatGPT to assess its determinism in generating an answer to a request: when the input is partially specified or incomplete, thus not adequately limiting the solution space, ChatGPT generates several different answers, often completely wrong or not suitable to the request. Therefore, our layer should have provided ChatGPT not only with the user's sentence and the generic request of generating an XML file describing a robot task, but also with a set of additional specifications to make the system converge toward a unique and correct solution.

After these preliminary tests, we derived that our layer must provide rigorous instructions to ChatGPT about the structure of the XML files. To this end, we decided to provide the system with a dataset of XML files created in previous experiments with CAPIRCI as training examples for learning the correct structure of an XML file describing a robot task, that is, learning the XML Schema Definition (XSD). Also in this case, since the dataset did not cover a complete suite of cases, the output files generated by ChatGPT resulted to be very different one another (non determinism) and sometimes inconsistent.

A third step was creating a request to ChatGPT that combines i) the XSD obtained in the previous step that was able to validate the highest percentage of XML documents, with ii) a set of constraints (still expressed in natural language) that better clarify the roles of tags and attributes to be used for generating the XML files describing the robot tasks. A convergent behavior and correct XML documents were obtained in this case, allowing us to delineate a new architecture of CAPIRCI, which integrates a new NLP layer with an *adapter* able to invoke ChatGPT APIs for user's request interpretation (see Figure 3).

Figure 4 shows an example of request by the adapter to ChatGPT to interpret the user's sentence "Pick up ten flasks, rotate each flask twice, and put them in the container" (corresponding to the dialogue with our previous chat shown in Figure 1). The adapter functionality is fundamental to limit the creativity and non determinism of ChatGPT.

With this approach the natural language interaction may become even more natural than that offered by the original chat-based interface of CAPIRCI. In fact, users can express their requests with unique and articulated sentences without being involved in a rigid exchange of speech turns for gathering all information necessary to generate a complete task description.

## 5. Discussion and Conclusion

In this paper, we have explored how to exploit the NLP capabilities of OpenAI ChatGPT to generate robot programs. Differently from existing proposals (e.g., [14]), our aim was not to

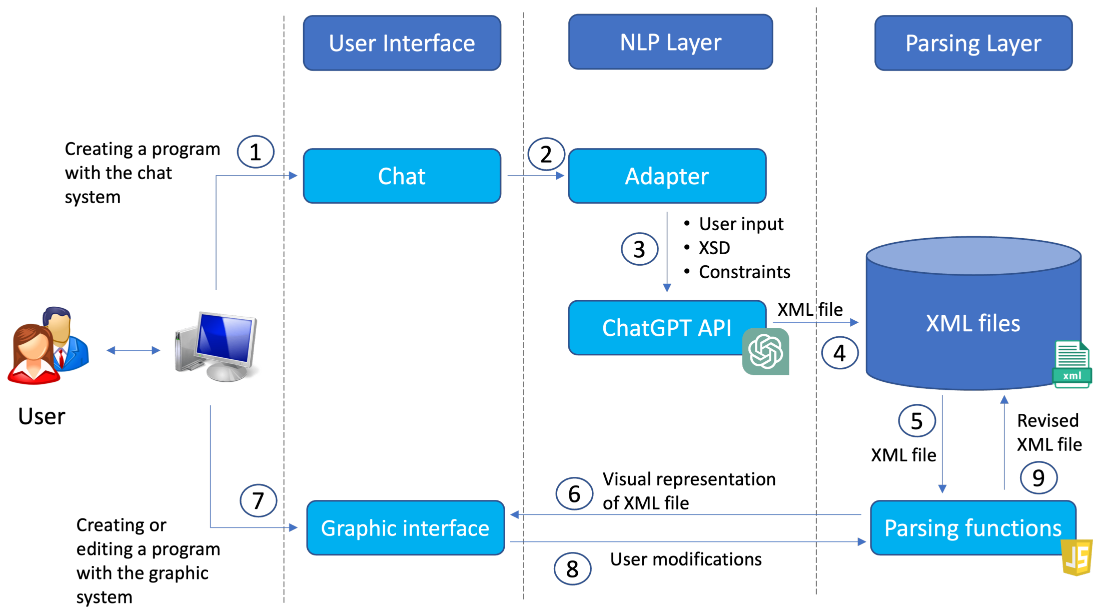


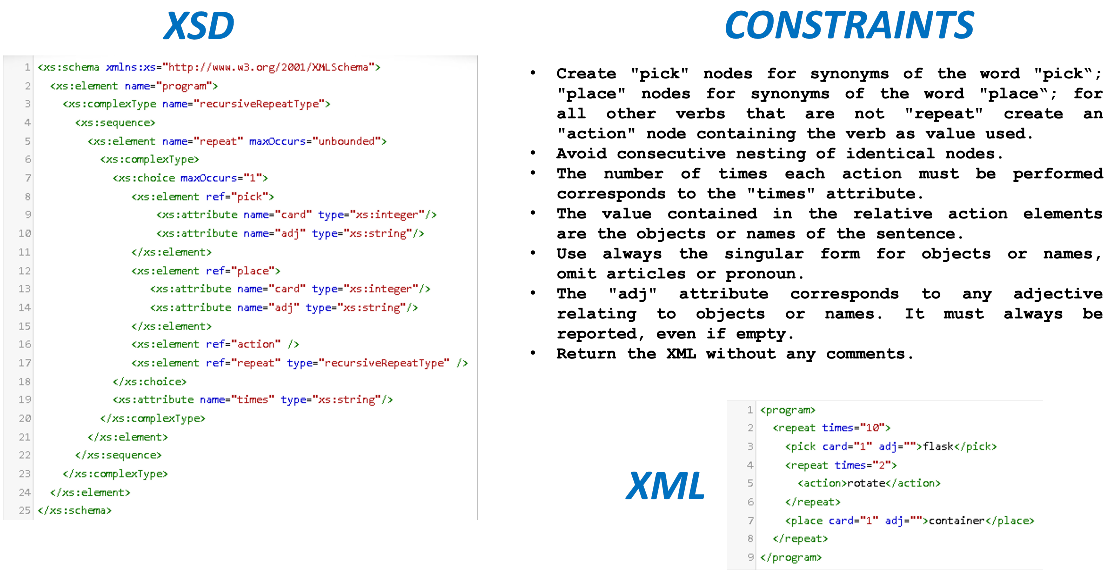
**Figure 3:** The new architecture of CAPIRCI integrating ChatGPT: the numbers indicate the workflow when program creation starts with using the chat.

generate the final code for the collaborative robot starting from the user's request, but rather to obtain an intermediate representation of the program that could be processed in our EUD environment CAPIRCI to provide a user, neither expert in programming nor in robotics, with the possibility of verifying the correctness of the generated program.

In this way, we allow the user to keep on expressing their requests for robot programming using an intuitive interaction through the formulation of sentences in natural language based on the concepts of the domain. On the other hand, we provide the user with the block-based representation of the robot program corresponding to their request, as was originally in CAPIRCI [6, 7], in order to assess its correctness and completeness, and tune it as needed, by simply manipulating the visual blocks.

In synthesis, the idea is to use the Artificial Intelligence features of ChatGPT as services for advanced natural language understanding, leaving the control on the final robot program description to the human, in a way that the human can find simple and intuitive. Indeed, the target users of CAPIRCI are not software programmers, but workers that, at some time during the usage of a collaborative robot, may need to program a new robot task to accomplish a specific work. In this regard, CAPIRCI can become a EUD environment with a high degree of EUDability [15] also for workers with a low level of computational fluency. In addition, computational fluency may be nurtured by the system helping users to gradually acquire the concepts of abstraction, decomposition, algorithm design, generalization, and evaluation.

The approach presented in this paper is very preliminary and has several limitations. The most severe one is that only the generation of XML descriptions of small and simple tasks for a restricted domain has been tested. Thus, further experimentation and tuning of the system are needed. Moreover, another type of system extension (still a EUD activity) was implemented



```xml
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="program">
    <xs:complexType name="recursiveRepeatType">
      <xs:sequence>
        <xs:element name="repeat" maxOccurs="unbounded">
          <xs:complexType>
            <xs:choice maxOccurs="1">
               <xs:element ref="pick">
                   <xs:attribute name="card" type="xs:integer"/>
                   <xs:attribute name="adj" type="xs:string"/>
               </xs:element>
               <xs:element ref="place">
                  <xs:attribute name="card" type="xs:integer"/>
                  <xs:attribute name="adj" type="xs:string"/>
               </xs:element>
               <xs:element ref="action" />
               <xs:element ref="repeat" type="recursiveRepeatType" />
            </xs:choice>
            <xs:attribute name="times" type="xs:string"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

- Create "pick" nodes for synonyms of the word "pick"; "place" nodes for synonyms of the word "place"; for all other verbs that are not "repeat" create an "action" node containing the verb as value used.
- Avoid consecutive nesting of identical nodes.
- The number of times each action must be performed corresponds to the "times" attribute.
- The value contained in the relative action elements are the objects or names of the sentence.
- Use always the singular form for objects or names, omit articles or pronoun.
- The "adj" attribute corresponds to any adjective relating to objects or names. It must always be reported, even if empty.
- Return the XML without any comments.

```xml
<program>
  <repeat times="10">
    <pick card="1" adj="">flask</pick>
    <repeat times="2">
      <action>rotate</action>
    </repeat>
    <place card="1" adj="">container</place>
  </repeat>
</program>
```

**Figure 4:** XSD and constraints provided to ChatGPT for the running example and XML document generated as output.

in the original version of CAPIRCI [7]: it supported users to enrich the natural language and graphic blocks by defining new objects, actions, and locations to be used in the description of robot tasks; this was performed through the interaction with the graphical interface of CAPIRCI and image processing algorithms. We plan to integrate in the future the use of ChatGPT also for this EUD activity.

## References

[1] V. Villani, F. Pini, F. Leali, C. Secchi, Survey on human–robot collaboration in industrial settings: Safety, intuitive interfaces and applications, Mechatronics 55 (2018) 248–266. doi:https://doi.org/10.1016/j.mechatronics.2018.02.009.

[2] J. Huang, M. Cakmak, Code3: A system for end-to-end programming of mobile manipulator robots for novices and experts, in: Proceedings of the 2017 ACM/IEEE International Conference on Human-Robot Interaction, HRI '17, Association for Computing Machinery, New York, NY, USA, 2017, p. 453–462. doi:10.1145/2909824.3020215.

[3] C. Schou, R. S. Andersen, D. Chrysostomou, S. Bøgh, O. Madsen, Skill-based instruction of collaborative robots in industrial settings, Robotics and Computer-Integrated Manufacturing 53 (2018) 72–80. doi:https://doi.org/10.1016/j.rcim.2018.03.008.

[4] C. Paxton, F. Jonathan, A. Hundt, B. Mutlu, G. D. Hager, Evaluating methods for end-user creation of robot task plans, in: 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2018, pp. 6086–6092. doi:10.1109/IROS.2018.8594127.

[5] G. Fischer, Computational Literacy and Fluency: Being Independent of High-Tech Scribes,

Hildesheim, 2005, pp. 217–230. URL: https://l3d.cs.colorado.edu/wordpress/wp-content/uploads/2016/04/hightechscribes-05.pdf.

[6] S. Beschi, D. Fogli, F. Tampalini, Capirci: A multi-modal system for collaborative robot programming, in: A. Malizia, S. Valtolina, A. Morch, A. Serrano, A. Stratton (Eds.), End-User Development, Springer International Publishing, Cham, 2019, pp. 51–66.

[7] D. Fogli, L. Gargioni, G. Guida, F. Tampalini, A hybrid approach to user-oriented programming of collaborative robots, Robotics and Computer-Integrated Manufacturing 73 (2022) 102234. URL: https://www.sciencedirect.com/science/article/pii/S073658452100106X. doi:https://doi.org/10.1016/j.rcim.2021.102234.

[8] H. Lieberman, F. Paternò, V. Wulf, End User Development (Human-Computer Interaction Series), Springer-Verlag, Berlin, Heidelberg, 2006.

[9] F. Paternò, V. Wulf (Eds.), New Perspectives in End-User Development, Springer, Cham, 2017. doi:10.1007/978-3-319-60291-2.

[10] B. R. Barricelli, F. Cassano, D. Fogli, A. Piccinno, End-user development, end-user programming and end-user software engineering: A systematic mapping study, Journal of Systems and Software 149 (2019) 101–137. doi:https://doi.org/10.1016/j.jss.2018.11.041.

[11] N. K. Lincoln, S. M. Veres, Natural language programming of complex robotic bdi agents, Journal of Intelligent Robot Systems 71 (2013) 211–23. doi:https://doi.org/10.1007/s10846-012-9779-1.

[12] M. Stenmark, P. Nugues, Natural language programming of industrial robots, in: IEEE ISR 2013, 2013, pp. 1–5. doi:10.1109/ISR.2013.6695630.

[13] D. K. Misra, J. Sung, K. Lee, A. Saxena, Tell me dave: Context-sensitive grounding of natural language to manipulation instructions, The International Journal of Robotics Research 35 (2016) 281–300. doi:10.1177/0278364915602060.

[14] S. Vemprala, R. Bonatti, A. Bucker, A. Kapoor, ChatGPT for Robotics: Design Principles and Model Abilities, Technical Report, Microsoft, 2023. URL: https://www.microsoft.com/en-us/research/uploads/prod/2023/02/ChatGPT_Robotics.pdf.

[15] B. R. Barricelli, D. Fogli, A. Locoro, Eudability: A new construct at the intersection of end-user development and computational thinking, Journal of Systems and Software 195 (2023) 111516. doi:10.1016/j.jss.2022.111516.