Democratising Access to Logic Programming: A Web Application Design Tool for Querying Prolog Code

Santiago Andrés Villarroel[1,*], Christian Nelson Gimenez[1], Jorge Pablo Rodríguez[1] and Laura Andrea Cecchi[1]

[1]*Grupo de Investigación en Lenguajes e Inteligencia Artificial (G.I.L.I.A.)*
Facultad de Informática, Universidad Nacional del Comahue
Neuquén, Argentina

Abstract

Logic Programming (LP) is a powerful paradigm for gaining fundamental knowledge and skills in Computer Science. LP facilitates the development of Computational Thinking skills, which are relevant for problem-solving, and also strengthens Logical Thinking abilities. To teach LP effectively, specialised educational resources are necessary. While students develop their programs in Prolog, they often struggle to showcase their running applications to classmates, friends, and family.

Providing educational resources that support the creation of Web applications with Prolog querying functionality will motivate students to learn.

This work presents Prolog Web App Creator, an integrated development environment for students to design and create Web applications. Ease of use, collaboration, and publication of the result are also relevant aspects of the environment, allowing the students to share the design and the product with their social circles.

The proposed solution implements an educational resource to consolidate LP teaching while fomenting collaboration, democratisation, and strengthening current initiatives. Prolog Web App Creator empowers creative individuals to develop solutions using LP and encourages their shift from the role of technology consumers to that of technology creators.

Keywords

Logic Programming Education, Prolog, Web technology, Web application

1. Introduction and Motivation

Recently, there has been a great interest in the early incorporation of Computer Science (CS) into mandatory education. The worldwide scientific community recognises learning CS to be an effective means to develop Computational Thinking (CT) and Logical Thinking (LT) in children [1, 2, 3, 4, 5].

In this direction, Logic Programming (LP) is a useful paradigm for acquiring fundamental knowledge and skills in CS. Furthermore, LP facilitates the exploration and development of CT skills, including abstraction, decomposition, pattern recognition, and generalisation, which are crucial for problem-solving. Additionally, LP fosters LT abilities, aiding in the differentiation of valid arguments from fallacies and contradictions, and in forming connections between arguments via reasoning, among other skills [6, 7, 8, 9, 10, 11].

To facilitate the effective teaching of LP, appropriate educational resources are necessary. In this regard, it is essential to not only visually make programs in Prolog, which avoids the need to teach and handle the textual syntax of this language, but also to query these programs in a user-friendly manner. However, this alone is not sufficient.

Students develop their programs in Prolog; nevertheless, they frequently lack the means to effectively show their running applications to their classmates, friends, and family. A step forward would be to allow any person to access and query the program. Providing educational resources that enable the

PEG 2024: 2nd Workshop on Prolog Education, October, 2024, Dallas, USA.
*Corresponding author.
santiago.villarroel@fi.uncoma.edu.ar (S. A. Villarroel); christian.gimenez@fi.uncoma.edu.ar (C. N. Gimenez); j.rodrig@fi.uncoma.edu.ar (J. P. Rodríguez); lcecchi@fi.uncoma.edu.ar (L. A. Cecchi)
0009-0008-8687-0619 (S. A. Villarroel); 0000-0002-8347-2526 (C. N. Gimenez); 0000-0002-4697-6477 (J. P. Rodríguez); 0000-0001-5236-6715 (L. A. Cecchi)

creation of Web applications with Prolog program querying functionality would motivate students to learn.

Computational devices like smartphones and personal computers have become integral to our daily lives. They are not merely tools for communication; they are now extensively utilised across various domains, including navigation, personal identification, gaming, and social networking. Their powerful computing capabilities, compact size, internet connectivity, affordability, and wide range of user-friendly apps make them highly usable.

Children and young people are keen technology consumers, especially when it comes to mobile technologies. The education sector is increasingly integrating these technologies, emphasising the importance of understanding their mechanisms to foster the development of technology creators. In this regard, it is crucial to make continuous efforts to develop new technological environments that enable the consolidation of LP in primary and secondary schools, increasing and strengthening existing initiatives.

In this context, our proposed solution consists of a Web-based educational resource that allows the creation of a user-customised Web application to query an already-built Prolog program. *Prolog Web App Creator* is the name of this tool and it is designed for novice users and non-programmers.

In the design and implementation of this resource, priority is given to ensuring that the user interface is highly intuitive so that its use does not require prior knowledge.

By using this tool, we aim to reach a wider range of students in LP education, providing an intuitive design environment that allows any user, especially children, to build situated Web applications. In this way, the goal is to democratise software development, empowering everyone, particularly school-aged children and youth, and facilitating their transition from the role of technology consumers to that of technology creators.

2. Related Works

In this section, we outline the tools and software that are relevant to the proposed work.

MIT App Inventor is a visual and intuitive programming environment that facilitates the development of mobile applications for everyone. The user interface consists of a *Designer* to select the components of the designed application and their properties, and a *Blocks Editor* to define the behaviour of the application [12].

The original design of Scratch was motivated by the needs and interests of young people. *Scratch* is a visual programming environment that allows users to learn programming by creating personally meaningful projects (i.e. motivated by the needs and interests of young people), such as games or stories [13].

Its main objective is to facilitate and promote self-directed learning through exploration, collaboration, and sharing with peers.

Snap! (formerly known as *BYOB*) is a visual, drag-and-drop programming language based on blocks [14]. It is an extended implementation of *Scratch* that allows managing more complex data structures and has a more youthful design. This extended functionality makes the tool suitable for a more in-depth introduction to computer science for secondary and university-level students.

Ciao Playground [15] is a platform that allows access to a Prolog engine embedded in the Web browser, without any need for prior software installation. Additionally, existing Prolog tutorials can be adapted for the Playground, or new ones can be created, to achieve a higher level of interactivity by allowing the user to test and manipulate code on the Web. Similarly, other Prolog implementations also have Web tools that allow interaction with a Prolog engine, such as SWI-Prolog with the SWISH [1] platform and Tau Prolog Sandbox[2]. However, unlike our proposal, these tools do not generate an application nor include an educational visual component.

[1] https://swish.swi-prolog.org/ visited on August 17, 2024
[2] http://tau-prolog.org/sandbox/ visited on August 17, 2024

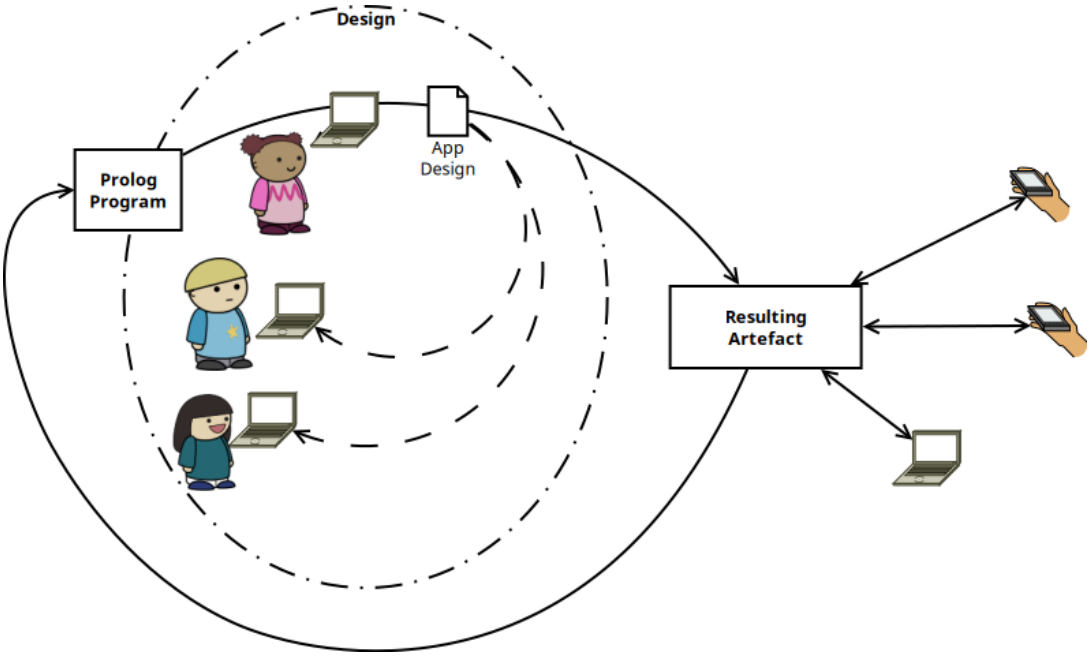

Figure 1: Resulting Artefact life cycle.

Blockly Prolog[3] is a Web-based visual educational resource that facilitates the creation of a Prolog program using blocks. The tool allows inserting blocks corresponding to concepts such as facts, constants, and variables, into a Canvas to construct a Prolog program. This functionality achieves a higher level of accessibility by not requiring the handling of Prolog textual syntax. Although this tool has a strong visual component, it is not focused on creating an application that interacts with a given Prolog program, nor can it be shared with an audience unfamiliar with LP.

MIT App Inventor, Scratch, and Snap! are powerful tools that enable non-programmers to build and share their applications, which is the core functionality that Prolog Web App Creator aims to achieve. These platforms, however, are rooted in the Imperative Programming Paradigm, which differs from the LP approach that the proposed tool adopts. While the paradigms vary, the overarching objective remains the same, that is, to further democratise access to programming, expanding the reach of this discipline to a broader audience.

In contrast, tools like Blockly Prolog, SWISH, Tau Prolog Sandbox and Ciao Playground also contribute to the goal of increasing the accessibility of the knowledge area of programming, especially LP. However, their focus differs from that of the previously mentioned tools, as they do not aim to design, build, and share executable Web applications. Instead, these tools emphasise providing interactive environments for learning and exploring LP concepts.

3. How Prolog Web App Creator works

Prolog Web App Creator is an integrated development environment (IDE) designed for beginners and non-programmers to create Web applications with features like querying Prolog programs. These Web applications, which we will refer to as the Resulting Artefact, can be used and shared by anyone in the students' social circles, such as family and friends.

Children and teenagers can develop software using their own Prolog code or by remixing code from their peers, thus embracing the role of creators rather than merely being technology consumers.

Figure 1 illustrates the Resulting Artefact life cycle, which begins with a Prolog program. The program, created by the student or another person, is used as input for the IDE. The students create

[3] http://www.programmierkurs-java.de/blocklyprolog/editor/index.html visited on August 17, 2024

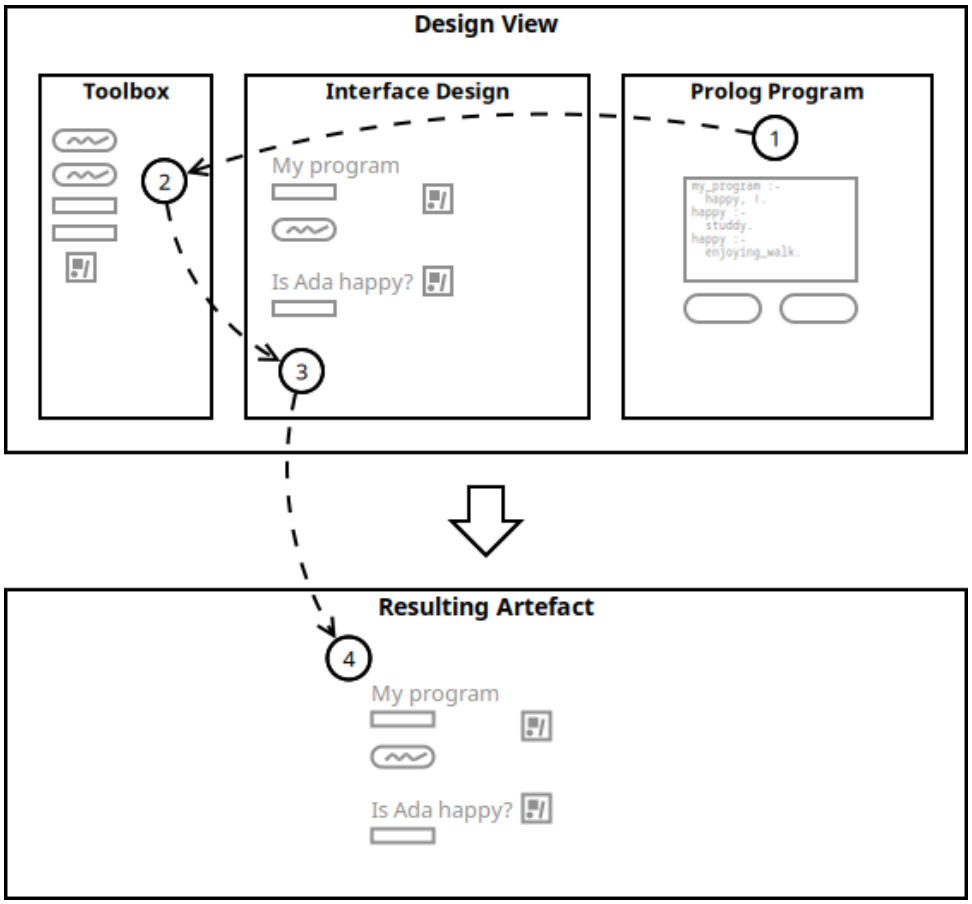

Figure 2: Prolog Web App Creator interface structure

their desired interface, populating the design with text, images, and dynamically generated elements based on the input program. The design can also be shared with their classmates to create different versions of the resulting application.

When the design is finished, the application is saved, and a Resulting Artefact that becomes accessible from the Internet is generated, allowing students to share it with other people.

Figure 2 illustrates the typical flow, along with the Prolog Web App Creator interface structure. The interface consists of two views: the design view and the resulting artefact view. The design view allows the student to create the application by providing a Prolog program as input (1) and inserting buttons (2) and images on a canvas (3). In this view, students produce their artefact and test the logic program before generating the final application. The resulting artefact view (4) shows the designed interface and executes the code by pressing the buttons placed by the student.

Figure 3 illustrates a project in the design view, where the input Prolog program is displayed in the right panel, representing a knowledge base about people and their respective hobbies. To create a Web application, first, the code is uploaded to the interface (see code panel on the right side in Figure 3). This action creates buttons automatically: one for each predicate in the uploaded logic program (see *Predicados* in Figure 3). To facilitate the design and configuration of the queries to the Prolog program, each button corresponding to a predicate not only includes its name but also indicates its arity. Note that a small square appears for each argument in the predicate. For example, the relation *persona* is unary and the relation *hobby* is binary. These buttons are associated with pre-defined elements called Widgets.

Then, the student can press these buttons associated with various types of Widgets, such as images, text, or queries. This action will result in these elements being inserted into the canvas, creating their personalised application. The Toolbox (see Figure 2; left panel in Figure 3) contains Query Widgets,

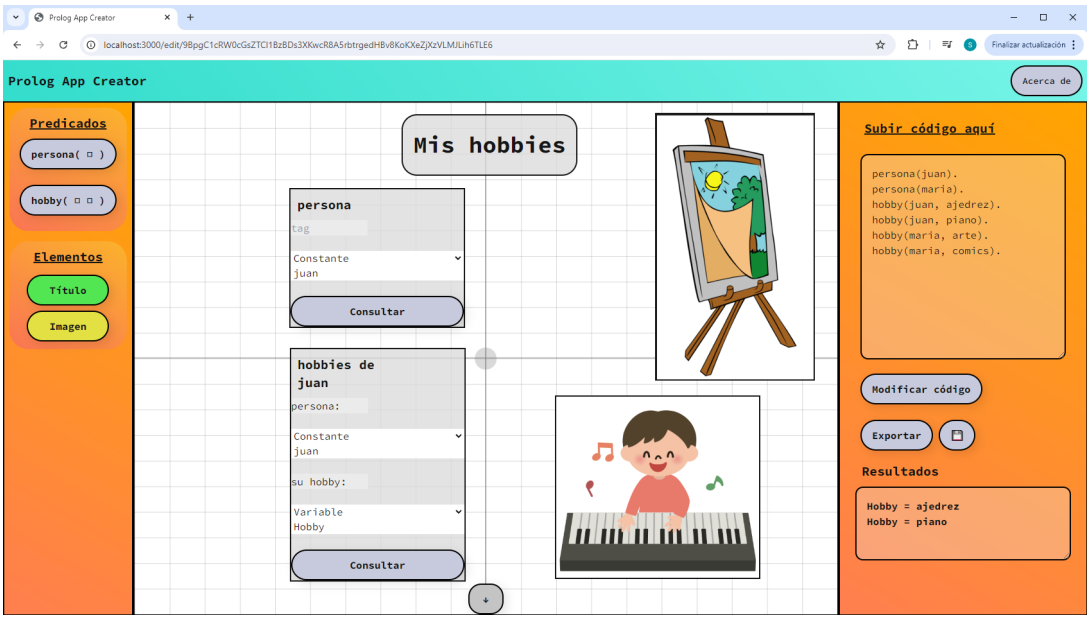

Figure 3: Prolog Web App Creator interface in the editing view.

which were generated based on the predicates present in the input program. On the Canvas, a Title, two Images, and two Query Widgets have been inserted. The Query Widgets consist of a title, an optional tag, parameter type selectors, and the parameters themselves. Interacting with these Widgets by pressing the *Consultar* (Query) button will display the results in the lower right panel. Once the student is satisfied with their design, they generate the Resulting Artefact (see *Exportar* button in Figure 3).

The tool structure consists of three main components: Client, Prolog Engine, and Server. The Client component will handle all the aspects related to the GUI (Graphical User Interface), and the interaction with the Prolog Engine. In particular, the Client holds the primary responsibility for managing the views explained before, and the design and editing process of the Resulting Artefact. In addition, it is equipped to communicate user-generated queries via Widgets to the Prolog engine, ensuring seamless interaction and retrieval of the results. The Server, on the other hand, handles the persistent storage of user-generated projects. It is designed to save these projects consistently and can retrieve them upon request from the Client. It is important to note that the Client component is equipped with a Prolog Engine, specifically the WebAssembly implementation of the Ciao Prolog engine [15]. This means that the Client does not rely on a network connection to the engine, thus achieving a higher level of availability.

4. Prolog Web App Creator in Primary and Secondary School

We aim to make LP accessible to students without a deep disciplinary background. Thus, we seek to broaden the representation of CS in primary and secondary education. There is a significant disparity and a lack of a clear definition of how CS is integrated into the curriculum at different educational levels. Furthermore, CS is entirely absent from the curriculum at several of these educational levels [3, 16].

The purpose of this work is to democratise access to computing and LP in primary and secondary education, rather than limiting it to higher education. LP is not a common topic in schools despite its relevance. Usually, teachers do not have the necessary training to teach LP without the aid of a specialised tool. Therefore, textual codification should not be a starting requirement. Thus, Prolog Web App Creator opens up paths to a first contact with LP not focused on textual programming.

This is particularly relevant for teachers who need to develop content knowledge and pedagogical content knowledge for effectively teaching LP. Teachers dealing with teaching LP need tools and

strategies specifically designed for this purpose.

A constructionist approach, where individuals focus on building tangible and public artefacts, is particularly effective in giving learners a sense of purpose when engaging with LP. This approach encourages students to create meaningful projects, making abstract concepts in LP more concrete and applicable. From a constructionist perspective, there is a strong emphasis on interaction, where the student consciously participates in the construction of a public entity [17, 18]. Within the scope of this work, a public entity refers to the Resulting Artefact, which is a computational artefact that performs queries on a Prolog-based knowledge base.

We identify two primary ways in which students can interact with this paradigm using the tool. The first option entails working with a Prolog program that is not of our authorship. In this scenario, the student learns to read and interpret an existing knowledge base to build an application. The student can then deepen their understanding by modifying and extending the existing program, allowing them to apply their knowledge of LP and Prolog concepts in a hands-on manner [19].

The second option involves the student building an application based on their own Prolog program. This approach provides the learner with a meaningful resource that can be interacted with by others, reinforcing the practical application of LP principles.

Under this scenario, teachers and students select a problem on the curriculum, conduct research, and define its scope. For example, environmental problems. In this direction, they could identify environmental issues and determine pollutants and their effects on the environment and health. Teachers and students could collaboratively work to create a text on the topic (an example in Spanish is available at this link). They start by defining and coding facts about this topic and then create rules that describe relationships between facts and a possible solution. Students in primary school could make the program

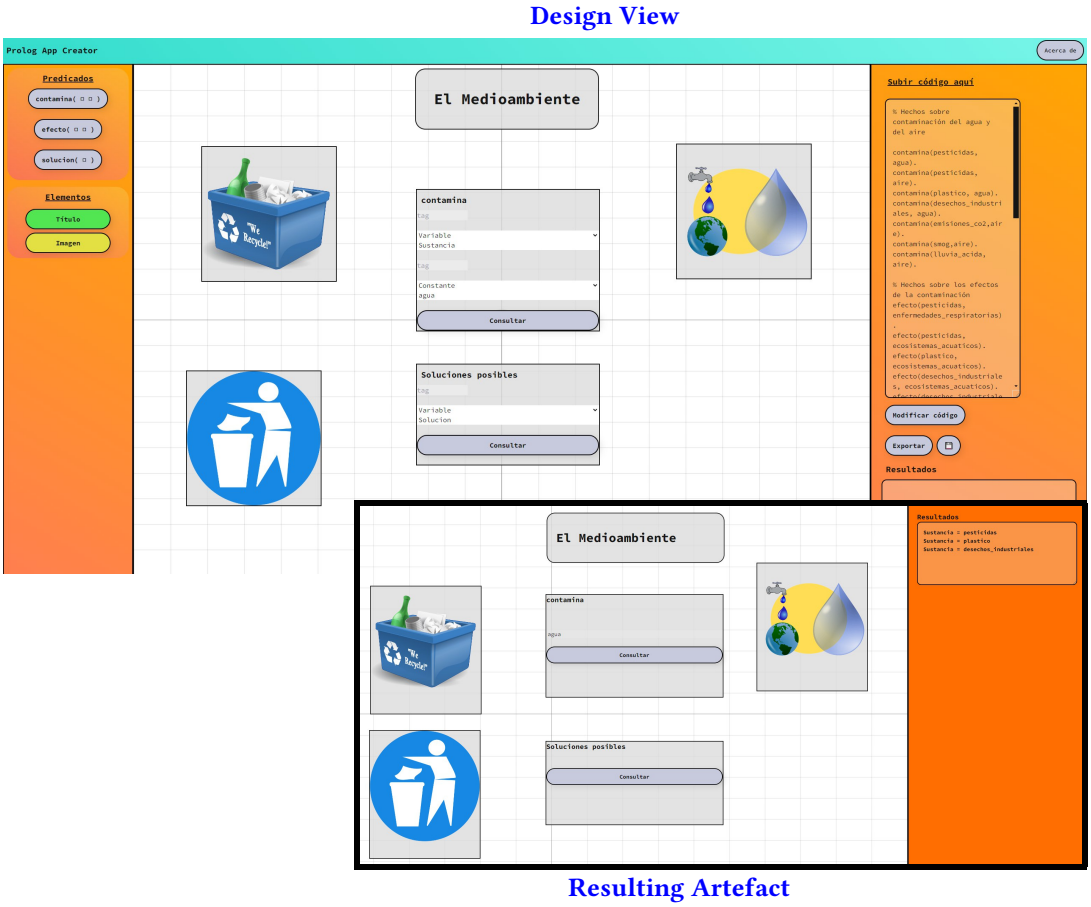

Figure 4: Designing a Web app for environmental problems.

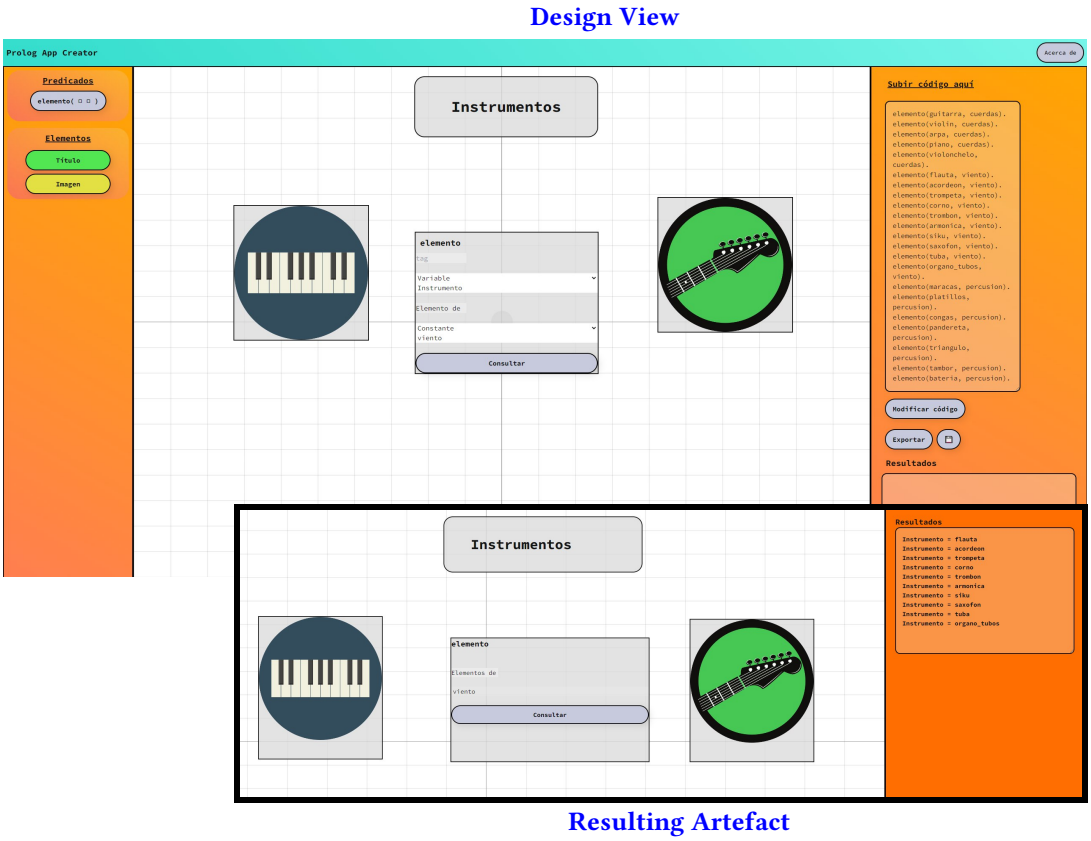

Figure 5: Designing a Web app for classifying instruments.

in Blockly Prolog (see block-based logic program) and students in secondary school could do it in textual Prolog (see textual logic program). Once students have created their logic program, they upload the code into Prolog Web App Creator, design the Web app, and export the Resulting Artefact (see Figure 4).

Similarly, a primary school music teacher, Juan Matías Fernández, who teaches at Escuela Primaria 125 “Rosalía Núñez de Alcaraz” in Neuquén, Patagonia, Argentina, design a class for his students who are 8 or 9 years old. He presents a classification for instruments: string, percussion and wind (the block-based logic program can be accessed in this link and the textual logic program is available in this link to Ciao Prolog Playground). Figure 5 shows the Design View and the Resulting Artefact created.

Prolog Web App Creator was designed to easily create a personalised Web application using a Prolog code as input, by inserting graphical pieces on a canvas. This allows children and young students to quickly build prototypes for their ideas, test them, and make improvements. Furthermore, the created public entity can be shared with their family and friends.

5. Conclusions and Future Work

In this work, we propose Prolog Web App Creator, an integrated environment that enables users to create their own Web applications for querying an input logic program using specified widgets and related functionality. It allows the created Web application to be exported to a final view (referred to as the *Resulting Artefact*), which can be shared via a link and access it with a Web browser. Prolog Web App Creator facilitates the practice of basic LP and Prolog concepts while encouraging creativity that results in a meaningful and personalised final product for the user.

The alpha version of the Prolog Web App Creator is available at https://prologforkids.fi.uncoma.edu.ar/.

Overall, the proposed solution implements a visual and Web-based educational resource, to use as a

new technological environment to consolidate LP while increasing participation. The path taken by environments, such as Scratch, provides consistent evidence that these types of tools play an important role in empowering people, democratising CS and improving learning. Within this framework, it is expected that Prolog Web App Creator has ample opportunities to contribute to expanding the presence of LP in primary and secondary education.

Based on this work, the following lines for further research and development are proposed:

- Expand aspects related to creativity and customisation in the design process of the Resulting Artefact.
- Translate the tool into other languages, increasing its target audience.
- Incorporate several examples of pre-designed Prolog applications, as supporting guides or as starting points, to illustrate the various functionalities of the tool.
- Extend the tool to support block-based Prolog programs, in order to avoid contact with textual syntax.

References

[1] S. Combéfis, G. Beresnevičius, V. Dagienė, Learning programming through games and contests: overview, characterisation and discussion, Olympiads in Informatics 10 (2016) 39–60.

[2] J. Moreno-León, M. Román-González, G. Robles, On computational thinking as a universal skill: A review of the latest research on this ability, in: 2018 IEEE Global Engineering Education Conference (EDUCON), 2018, pp. 1684–1689. doi:10.1109/EDUCON.2018.8363437.

[3] S. Bocconi, A. Chioccariello, P. Kampylis, V. Dagienė, P. Wastiau, K. Engelhardt, J. Earp, M. A. Horvath, E. Jasutė, C. Malagoli, et al., Reviewing Computational Thinking in Compulsory Education, Technical Report, Joint Research Centre (Seville site), 2022.

[4] R. Society, After the reboot: Computing education in uk schools, Policy Report (2017).

[5] P. Kampylis, V. Dagienė, S. Bocconi, A. Chioccariello, K. Engelhardt, G. Stupurienė, V. Masiulionytė-Dagienė, E. Jasutė, C. Malagoli, M. Horvath, et al., Integrating computational thinking into primary and lower secondary education, Educational Technology & Society 26 (2023) 99–117.

[6] S. Beux, D. Briola, A. Corradi, G. Delzanno, A. Ferrando, F. Frassetto, G. Guerrini, V. Mascardi, M. Oreggia, F. Pozzi, et al., Computational thinking for beginners: A successful experience using prolog., in: CILC, 2015, pp. 31–45.

[7] V. Tabakova-Komsalova, S. Stoyanov, A. Stoyanova-Doycheva, L. Doukovska, Prolog education in selected high schools in bulgaria, in: D. S. Warren, V. Dahl, T. Eiter, M. Hermenegildo, R. Kowalski, F. Rossi (Eds.), Prolog - The Next 50 Years, number 13900 in LNCS, Springer, 2023.

[8] Y. Zhang, J. Wang, F. Bolduc, W. G. Murray, LP based integration of computing and science education in middle schools, in: Proceedings of the ACM Conference on Global Computing Education, 2019, pp. 44–50.

[9] T. T. Yuen, M. Reyes, Y. Zhang, Introducing computer science to high school students through logic programming, Theory and Practice of Logic Programming 19 (2019) 204–228.

[10] L. A. Cecchi, J. P. Rodríguez, V. Dahl, Logic programming at elementary school: Why, what and how should we teach logic programming to children?, in: D. S. Warren, V. Dahl, T. Eiter, M. Hermenegildo, R. Kowalski, F. Rossi (Eds.), Prolog - The Next 50 Years, number 13900 in LNCS, Springer, 2023.

[11] J. Rodriguez, L. Cecchi, Logic programming in primary school: Facing computer science at an early age, Proceedings of 50 Conferencia Latinoamericana de Informática (CLEI) (2024).

[12] S. C. Pokress, J. J. D. Veiga, Mit app inventor: Enabling personal mobile computing, arXiv preprint arXiv:1310.2830 (2013).

[13] J. Maloney, M. Resnick, N. Rusk, B. Silverman, E. Eastmond, The scratch programming language and environment, ACM Transactions on Computing Education (TOCE) 10 (2010) 1–15.

[14] B. U. of California, Snap!, https://snap.berkeley.edu/about, ???? Accessed: April 04, 2024.

[15] G. García Pradales, Monaco playground for ciao prolog, 2022. URL: https://oa.upm.es/71073/, trabajo Final de Grado.
[16] J. Rodríguez, M. M. Cortez, S. Boari, Exploration of the place of computer science knowledge areas in the argentine secondary school: A systematic review, Electronic Journal of SADIO 21 (2022).
[17] I. E. Harel, S. E. Papert, Constructionism., Ablex Publishing, 1991.
[18] S. Papert, I. Harel, Situating constructionism, constructionism 36 (1991) 1–11.
[19] S. Sentance, J. Waite, M. Kallia, Teaching computer programming with primm: a sociocultural perspective, Computer Science Education 29 (2019) 136–176.