MissionEDU - Developing a Unified Educational Robotics Experience

Daniel Swoboda, Daniel Honies, Christoph Käferle, Markus Pinter, Florian Ungersböck, Raphael Weinfurter

Department for Computer Science, Secondary Technical College, HTL Wiener Neustadt

MissionEDU - Developing a Unified Educational Robotics Experience

I. INTRODUCTION

Computer programming, software development and robotics skills are on their way to becoming the most important abilities to have. With the introduction of computerized systems and automation the efficiency of most industries rose. However, these benefits often led to unemployment waves. More advanced knowledge of computer science will be required from future workers in order to survive in a mid 21st century industry. [1]

To counteract the growing demands for employees with computer-knowledge and engineers a variety of different educational robotics programs and educational programming courses were created by various institutions [2]. Established robotics programs like Botball or RoboCup are trying to reach out to younger audiences through special elementary school competitions while educational programming courses like code.org and Scratch focus on teaching basic programming-skills to students through the use of graphical programming languages (GPLs) [3-5].

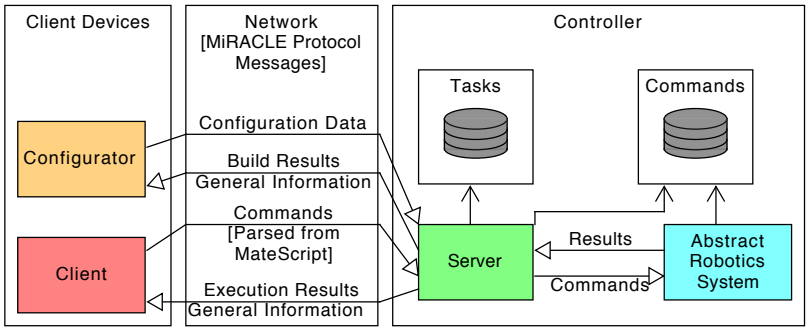

Fig. 1. The data flow between the sub-systems of MissionEDU

MissionEDU aims to combine these two concepts by creating a platform that is able to run on most modern robotics controllers. By doing so it enables the interplay of a fragmented eco-system, unifies the experience across devices and makes it easier to deploy graphical programming to existing hardware.

This publication introduces parts that are fundamental to the MissionEDU system and gives an overview of possible applications. The introduced parts include the Abstract Robotics System, the server, the programming client for tablet-PCs, the configurator client, the MiRACLE protocol, and the MateScript programming language. The way these subsystems interact is depicted in Figure 1.

II. ABSTRACT ROBOTICS SYSTEM

The abstract robotics system (ARS) is the system that enables MissionEDU to run on multiple controllers using the same code-base. Its design is based on the Robotic System Abstraction Layer (RSAL) developed for MissionControl. While the original RSAL made use of a XML-based language to auto-generate this platform specific code, the ARS of MissionEDU uses a JSON format to be consistent across all platforms. Another key difference is that the ARS is changeable during the runtime of the server by using on-demand recompilation. Where the RSAL parser generated C99 code, the ARS - by default - makes use of the modern C++14 standard and the features of C++. [6]

A. Concept

Since MissionEDU needs to access the robotics system of a controller in order to control the hardware, the server has to include platform specific code. Because of the multitude of different robotics controllers and the inconsistency of their firmware the server code would have to be changed for every platform. To separate the platform specific code from the server code and to allow the user to add their own methods the ARS was created. The ARS acts as a middleware through which the hardware is accessed and is therefore comparable to the hardware abstraction layer of modern operating systems [7].

Because the methods provided in the client app are proxies of the compiled platform specific functions created by the users that are stored in the ARS, it also fits the criteria of a remote procedure call back-end.

B. ARS Communication Interface

ARS integrates with the server through the ARS communication interface (ARS-CI) - which is a minimal request-response protocol - using unix domain sockets (UDS) as the medium. The protocol consists of 5 different messages, 2 of them are used for connection management, the other 3 are used to control the ARS. The messages are transmitted ASCII formatted and line-wise.

C. Implementation

The ARS code is generated from a C++ base program - that implements the ARS-CI logic - and platform specific code necessary to include the robotics system of the target platform and user defined functions (UDFs). These UDFs can be added by the user using the configuration application. Through it, methods designed for the students' tasks can be added.

D. Integration with Server

In order to make the ARS swappable during run-time the server manages it as a subprocess. It is shut down when the configuration client connects. After the configuration client disconnects the ARS is restarted and normal operation is resumed. During configuration the ARS can be changed by the user. If the change leads to a non-compilable state it is reverted.

III. NETWORKING PROTOCOL

A. swockets Socket Wrapper

All parts of MissionEDU use a socket wrapper library - called swockets - specifically created for the project. swockets is available for all languages that are used as parts of MissionEDU: Python, C++ and C#. It uses the native socket APIs of every platform to wrap TCP connections. The wrapper is event-based and designed to only transmit JSON data. It automatically checks the data before transmission and includes methods for error checking and automated message stitching. swockets supports both a server and a client mode. In the server mode it automatically takes care of client connection management. While message receiving is by default asynchronous, it can be switched to a synchronous mode. [8]

Through the usage of self defined handlers the used protocol can be easily implemented. [8]

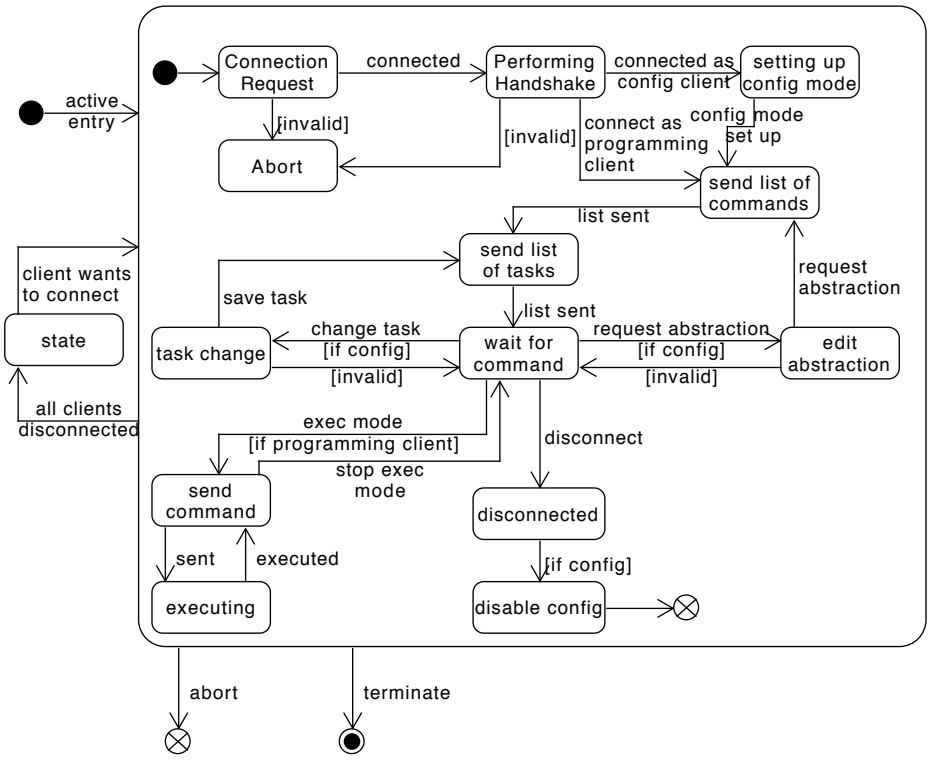

Fig. 2. A state transition diagram of the MiRACLE protocol

B. MiRACLE Protocol

In order to guarantee a controlled flow of data and de- fine the message formats the MissionEDU Robot Access, Configuration Loading and Enhancing Protocol (MiRACLE protocol) was created. MiRACLE is used for every network based communication between the parts of MissionEDU.

1) Description: The protocol is state-based and differentiates between a configuration client and a programming client and provides different functionality on both platforms. After a successful handshake in which the conditions of the connection are negotiated the server sends - in both modes - a list of all the tasks and all the functions available. In configuration mode the ARS is shut down and the actions to add, delete and change both functions and tasks are enabled. In programming mode control commands can be sent by the client which are forwarded to the server and executed by the ARS. A state transition diagram of the protocol can be seen in Figure 2.

2) Data Format: MiRACLE uses JSON based messages with a consistent general structure and a different payload for every message type (example depicted in Listing 1).

Listing 1. General structure of a MiRACLE message

```
{
	"MessageType":"<type >",
	"Payload" : {
		<payload data for message type>
	}
}
```

IV. SERVER

MissionEDU-Server manages client connections, handles method execution and controls the generation and compilation of the ARS. Similarly to the MissionControl server it is implemented using Python 2.7 - a language available on most Linux systems. The server is the backend of both the programming interface and the configuration environment.

A. System Design

The MissionEDU server was designed with standard robotics controllers in mind. These often ship with a Linux operating system and specifications close to those of smartphones. [9-10]

A server running on a robotics controller should limit its resource usage to a minimum to not interfere with the robotics system, especially when these systems perform heavy tasks like computer vision. To accomplish that the server makes use of a passive design where it only performs actions on request. The outsourcing of the robotics system access to the ARS here shows another advantage: Because the execution of the robot functions is performed by the

abstract robotics system the server once again can stay passive waiting for the execution to be finished.

It is also designed to be easily installed by using only the standard libraries of C++ (ARS) and Python (server). This increases the number of potential target platforms.

B. Implementation

To implement the server a language was needed that is already available on most platforms, that has an extensive standard library and includes a socket interface. Because of the system design choices not only compiled languages but also parsed languages were considerable. Since Python meets all the criteria and is an ergonomic language it was selected. Version 2.7 was chosen because it's both stable and available on most platforms [11]. Although Python 3.x is more modern and still under active development, it is not included on the Link or Wallaby controller, which would result in additional installation efforts.

C. Performance

Since the server is intended to run on robotics controllers which are - although they are becoming increasingly more powerful - often underpowered, server performance was a key objective of the project. The target was for the server itself to add less than 5% of average CPU.

To analyze the performance two key indicators were chosen: CPU and memory usage. The tested platform was the KIPR Wallaby controller (720 MHz; 512MB RAM) used in the Botball competition since 2016. The test results were compared to a 2012 MacBook Pro (2.3 GHz; 8GB RAM) for reference. 3 different scenarios were tested multiple times on each device: Idle usage (graphical representation of the results depicted in figure 3), pseudo-average load usage and compile usage.

The results of the memory analysis show a constant usage of system memory on both systems. This implies that there is neither a memory leak nor any unwanted allocation of memory during idle time. The overall usage in percent was proportionally bigger on the Wallaby because it has less memory.

CPU performance (Figure 3) stayed constant on the reference machine. However, the behavior on the Wallaby was notably different. At launch the server had a CPU usage of 1.5%. This did not stay constant but rather decreased over time ending with 0.7% at the end of the experiment. One explanation for this behavior could be that this is a result of the scheduler, while it also could be a result of JIT compilation.

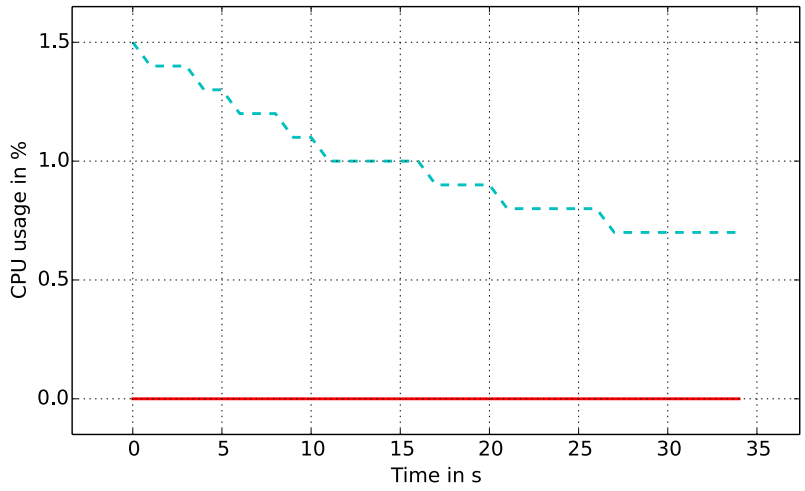

Fig. 3. Idle cpu usage on the Wallaby (blue, dashed) and reference machine (red)

V. CLIENTS

A. Programming Client

Tablet PCs and smartphones are common devices, found in European and American households [12]. Additionally educational usage of tablets is becoming more and more widespread with whole countries, including Austria, starting to give students access to these gadgets [13-14] . Because of that it was decided to create the MissionEDU client as a tablet app. To broaden the target audience a native multi-platform app (Android and iOS) with the option to extend it to the web was deemed to be necessary.

1) Client Concept and Task System: The concept of the app focuses on the idea of tasks. A task is a representation of an achievement a student has to accomplish. It consists of a description of what the student has to do and a list of UDFs the students can use. By selecting a task, the programming interface opens through which the program to solve the task can be created using the graphical blocks of the GPL. It is designed to reflect the sequence of a robotics course to make it easily integrable into the workflow of teachers and students.

Additionally there is free mode where students can create their own programs out of all methods that are available on the controller.

2) Design: From a design perspective the app uses bold colors and big icons (depicted in Figure 4) paired with a minimalistic user interface. These decisions were made to enhance readability and to make it more appealing to younger students.

The user interface is divided into three major parts with the first one being the log-in-screen, the second being the task-selection-screen and the third being the programming interface (PI).

In the PI students are displayed an expandable area to create their code by dragging graphical blocks which describe methods and flow-control structures and stitching them together.

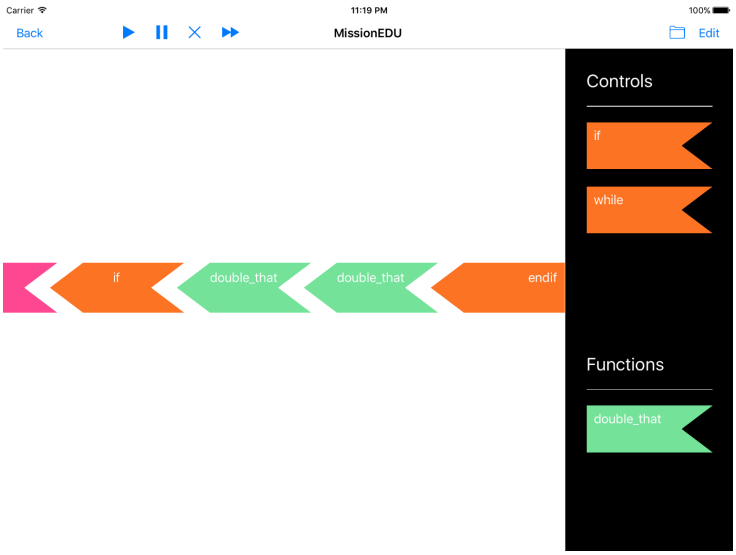

Fig. 4. A screenshot of the iOS Version in the programming interface view showing the different colors of the blocks.

3) Implementation: In order to reduce the effort of creating different apps for every platform and keep the possibility to use the same code-base for a web version React Native was chosen. Using this method one common code base can be used to write native apps for both Android and iOS using JavaScript as well as to develop a web-based version with the React Framework.

B. Configuration Client

The configurator or configuration client is the second type of client for MissionEDU. It is designed to be a desktop application through which the teachers are able to create and edit tasks as well as methods.

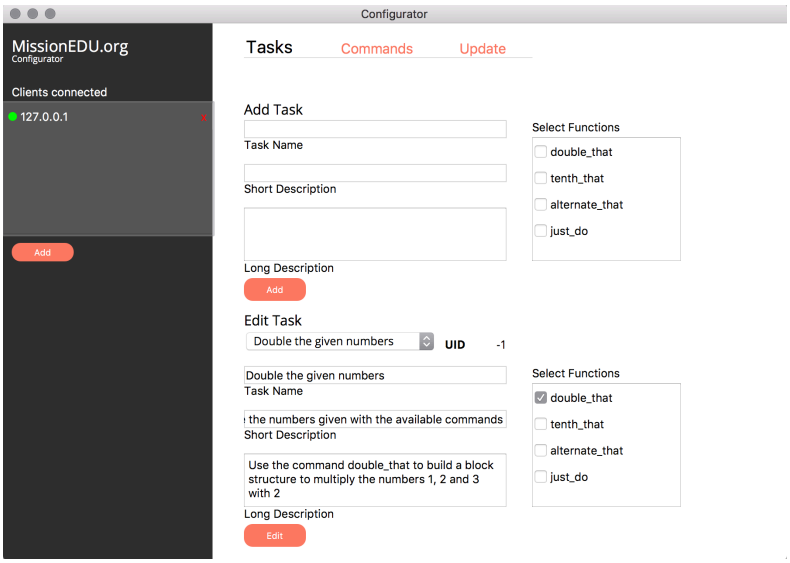

Fig. 5. The Configurator in the task view

1) Functionality: The configurator connects to the server using the C++ implementation of swockets. It is able to support multiple connections and features 3 views and a common side bar. In the side bar connections can be managed. Through the view ”Tasks” the tasks that are saved on the selected controller can be edited and new ones can be added (Figure 5). Via the ”Functions” view new methods can be added and old ones can be adapted. Changes are automatically saved on the controller and compiled to check for their functionality. If a method is not compilable the state is reverted.

MissionEDU configurator is a great improvement over the system of MissionControl where the user had to manually edit XML files on the controller. [6]

2) Implementation: The configurator was implemented in C++ using the UI toolkit Qt. Because of the way it is implemented the configurator at the time of writing supports macOS and most Linux distributions.

VI. MATESCRIPT

MateScript is the graphical programming language created from scratch for the MissionEDU project. It is at the time of publication not yet fully specified and functional and can therefore be considered a technology preview.

A. Language Description

MateScript is a graphical, functional, strictly typed programming language. Its components are described as graphical blocks with distinct functionality. MateScript consists of flow control blocks as well as parameterized and non-parameterized function calls. While the language doesn’t allow function creation using the graphical blocks, the provided functions can be used as

parameters or in flow control blocks as long as the data-types match. To reduce complexity only two levels of depth are supported. The blocks are stacked together horizontally.

The language supports the data types integer 32 bit, double precision and bool (according to the C++ specification)[15]. These datatypes can be used as parameters of function calls, in flow control structures or as return values. There is no support for variables.

Functions with one or no parameter are supported by MateScript. Each function call is represented by a distinct graphical block.

To enable flow control the control structures if, if-else, while and count are supported. Each of them is represented by a distinct graphical block. With if and if-else multiple function calls can be conditioned. With the count loop one or multiple function calls can be repeated n times. A while loop can be used to call one or multiple blocks as long as a condition is met.

B. Graphical Blocks

The graphical blocks of MissionEDU are colored blocks with an arrowhead like design that symbolizes the way they are stitched together.

For function calls a graphical block contains the name of the function, and optionally parameter selectors and space for other functions to be used as the parameters.

Control structure blocks are of different color and generally bigger than others. They consist of a starting and an ending block, between them are the blocks controlled by the structure. Parameters are included similarly to the function blocks.

C. Parsing and Execution

The graphical code created by the user is parsed into a syntax tree using the interpreter pattern. From there the code is executed sequentially. Each function call is transmitted to the server as a command where the ARS executes it. The results are sent back to the client and evaluated.

D. Comparison to other GPLs

Although existing and open languages could have been used, creating one specially for MissionEDU allowed for greater control and easier integration into the project since most graphical programming language libraries like Blocky or TUM.CMS.VPLControl are both platform specific and come with a pre-defined look and feel. Other graphical programming languages like the one used in Scratch or Code.org are application-bound and not really adaptable. MateScript - while mainly designed for usage within MissionEDU - is designed to be application agnostic, customizable in regards to the look and feel, and platform portable.

VII. APPLICATION IN EDUCATION

A. K12 Student Courses

The main focus of MissionEDU during both development and conception was its usage as an additional tool to be used in K12 robotics and programming beginner courses. In such an environment it can be used instead of the original programming interface to lower the entry barrier by providing a GPL. Students can then take the same hardware along when moving forward to a more advanced textual programming language.

B. Alternative Programming Interface

The project can also be used as an alternative programming interface to rejuvenate older controllers that might still be functional. Especially in public schools which often lack funding this allows the use of older yet functional hardware and supplies it with actively maintained software. Considering that through competitions like Botball - where every few years a new hardware generation is introduced - legacy hardware is created, MissionEDU could increase the amount of usable resources.

C. Controller Environment and Front-End

Since MissionEDU provides everything from a configuration tool to a programming interface it could theoretically be combined with custom firmware to create a controller software environment. Using the project could decrease the amount of work needed since all of the front-end software and a program execution engine would be already implemented.

D. Bridging a fragmented eco system

MissionEDU can also be used to combine a set of different controllers with similar specs and supply them with a unified programming interface. Through this the instructors' effort would be reduced by only having to explain one kind of programming interface and set of methods. Also money could be saved by not needing to update all the hardware at once but slowly replacing defunct controllers.

VIII. CONCLUSION

MissionEDU is a first step in creating a more unified experience and providing additional methods of programming to existing hardware. By making it possible to add graphical programming to multiple different controllers, old hardware can be repurposed and younger students can be taught more easily. It can also be beneficial to educational programs like the Junior Botball Challenge by providing an additional method of robot programming. In the future support for more programming languages for the ARS should be added. In addition to the graphical programming language a simple textual programming language should be provided. Also the system should be tested in cooperation with teachers and students of different schools

and ages to find ways of improving the workflow, the user interface, and the MateScript language.

APPENDIX

The MissionEDU source code is publicly available on http://github.com/MissionEDU, additional information is provided under www.missionedu.org.

ACKNOWLEDGMENT

The authors would like to thank Dr. Michael Stifter for his support during the work on this publication as project advisor.

REFERENCES

[1] R. Skidelsky *Rise of the robots: what will the future of work look like?* The Guardian
https://www.theguardian.com/business/2013/feb/19/rise-of- robots-future-of-work, 2013
[2] M. J. Mataric, N. Koenig, D. Feil-Seifer. *Materials for Enabling Hands- On Robotics and STEM Education*
AAAI Spring Symposium on Robots and Roboto Venues: Resources for AI Education, March 2007
[3] C. Stein *Botball: Autonomous students engineering autonomous robots*
https://peer.asee.org/botball-autonomous-students-engineering-autonomous-robots.pdf, accessed 2017-03-30
[4] H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, E. Osawa *RoboCup: The Robot World Cup Initiative*
http://dl.acm.org/citation.cfm?id=267738, accessed 2017-03-30
[5] D. Kumar Digital Playgrounds for Early Computing Education
acm Inroads, March 2014
[6] D. Swoboda, D. Honies, R. Weinfurter, M. Pinter, F. Ungersboeck and C. Kaeferle. *Remote Monitoring and Controlling of Robotic Systems with MissionControl*
European Conference on Educational Robotics, 2016.
[7] Katalin Popovici and Ahmed Jerraya. Hardware-dependant Software - Principles and Practice (Chapter 4)
Springer, 2009.
[8] D. Swoboda. *On Multiplatform TCP Socket Development*
swobo.space/multiplatform-networking-tcp-sockets, accessed 2017- 03-30.
[9] KIPR. *Wallaby Product page*
http://botballstore.org/product/wallaby- controller, accessed 2017-03-30.
[10] PRIA. *SCORE!*
https://pria.at/research/score/, accessed 2017-03-30.
[11] Python Foundation. *Python 2.7 Release*
https://www.python.org/download/releases/2.7/, accessed 2017-03-23.
[12] M. Sarwar, T. Soomro. *Impact of Smartphones's on Society*
European Journal of Scientific Research, Vol. 98 No 2 March, 2013, pp.216-226.
[13] M. Pegrum, C. Howitt, and M. Striepe. *Learning to take the tablet: How pre-service teachers use iPads to facilitate their learning*
Australasian Journal of Educational Technology, 2013, 29(4).
[14] Austrian Federal Ministry of Education. *Schule 4.0*
https://www.bmb.gv.at/schulen/schule40/index.html, accessed 2017- 03-30.
[15] cppreference.com. C++ Basic Concepts - Fundamental types
http://en.cppreference.com/w/cpp/language/types, accessed 2017-03- 30.