
DIPLOMARBEIT

MissionEDU - An Open Approach to Educational Robotics Systems

Ausgeführt im Schuljahr 2016/17 von:

Entwicklung Server, Konfigurationsprogramm
Daniel M. Swoboda 5AHIF-23

Entwicklung Client, Graphische Programmiersprache
Markus Pinter 5AHIF-19

Betreuer:

Dr. Michael Stifter

Wiener Neustadt, am 4. April 2017

Abgabevermerk:

Übernommen von:

Eidesstattliche Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche erkenntlich gemacht habe.

Wiener Neustadt, am 4. April 2017

Verfasser:

Daniel M. Swoboda

Markus Pinter

Contents

Eidesstattliche Erklärung	i
Contents	ii
Diplomarbeit Dokumentation	vi
Diploma Thesis Documentation	viii
Kurzfassung	x
Abstract	xi
Logo	xii
1 Introduction	1
1.1 Task	1
1.1.1 Features of MissionEDU	2
1.2 Visual Programming Languages	2
1.2.1 The History of Visual Programming Languages	2
1.2.2 Defining Visual Programming Languages	3
1.2.3 Specifying Visual Programming Languages	3
1.2.4 An overview of Educational Visual Programming Languages	3
1.3 The State of Educational Robotics	5
1.3.1 Competitions	6
1.3.2 Junior Competitions	7
1.3.3 Hardware used in Educational Robotics	7
1.4 Enabling Robotics Education through Visual Programming	7
2 Project Management	8
2.1 Kanban	8
2.1.1 Description of Kanban	8
2.1.2 History of Kanban	8
2.1.3 Application of Kanban in this project	8
2.2 Results	9
2.2.1 Schedule and content of meet-ups with the project advisor	9
2.2.2 Hours worked	9
3 Networking Technologies	10
3.1 Networking in MissionEDU	10
3.1.1 Data Flow	11

3.1.2	Data Encapsulation - Data Interchange Format	11
3.2	MiRACLE Protocol	11
3.2.1	Defining Protocols	12
3.2.2	Protocol Specification Theory	12
3.2.3	Protocol Design Theory	12
3.2.4	Transition and Sequence Specification of MiRACLE	13
3.2.5	Data Format and Message Specification of MiRACLE	15
3.2.6	MiRACLE Validation Model	15
3.3	Socket Wrapper Library swockets	16
3.3.1	Structure	16
3.3.2	Interface	17
3.3.3	Program Flow	18
3.3.4	Implementation of MiRACLE with swockets	19
4	Abstract Robotics Systems	20
4.1	A Description of Modern Educational Robotics Controllers	20
4.1.1	Raspberry Pi Setup	20
4.1.2	KIPR Wallaby Setup	21
4.1.3	Difference between Raspberry Pi and KIPR Wallaby	22
4.2	MissionEDU Abstract Robotics System	22
4.2.1	Hardware Abstraction Layers	22
4.2.2	ARS Design	23
4.3	Remote Procedure Calling with ARS	23
4.3.1	User Defined Methods (UDM)	23
4.3.2	ARS Communication Interface (ARS-CI)	23
4.3.3	Controller Definition File (CDEF)	24
4.3.4	ARS-GEN and Third Party Implementations	24
5	Server	25
5.1	Overview of Server-Client Systems	25
5.1.1	History of Servers	25
5.1.2	Modern Servers	26
5.1.3	Server-Client Systems in Robotics	26
5.2	Conception	26
5.2.1	General Design	26
5.2.2	Passive Design	27
5.2.3	Usage of ARS-CI and Choice of Language	27
5.2.4	Library Support and Ease-of-use of the Programming Language	27
5.2.5	Installation	27
5.3	Implementation	27
5.3.1	Programming Language	27
5.3.2	APIs and Libraries	28
5.3.3	Event Handling	28
5.4	Performance Analysis	28
5.4.1	Reasons for Analysis	28
5.4.2	Methods of Analysis	28
5.4.3	Tested Devices	29
5.4.4	Tested Indicators	29
5.4.5	Idle Performance	30
5.4.6	Regular Load Performance	31

5.4.7	Compile Performance	31
5.4.8	Discussion of Results	32
5.4.9	Methods to Increase the Performance	32
5.5	Compatibility	32
5.5.1	Minimum Requirements	32
5.5.2	Compatible Systems	32
5.5.3	Compatibility Issues	32
6	Configuration Client	33
6.1	Qt framework	33
6.2	Design and Functionality	34
6.3	Implementation	35
7	Robot Programming Interface	36
7.1	Overview of Robot Programming Interfaces	36
7.1.1	Botball Programming Interface	36
7.1.2	LEGO Mindstorms Programming Interface	37
7.1.3	Arduino Programming Interface	38
7.2	Design Strategies in Mobile Applications	38
7.2.1	Skeuomorphic Design	39
7.2.2	Flat Design	39
7.3	Graphical Element Design for VPLs	39
7.3.1	Vertical Linking	39
7.3.2	Horizontal Linking	39
7.4	MissionEDU Programming Interface	39
7.4.1	Programming Elements	40
7.4.2	Navbar	41
7.4.3	Programming Element Container	41
7.4.4	Workbench	41
7.5	Tasks	41
7.6	Programs	42
8	MateScript	43
8.1	Programming Language Design	43
8.1.1	Declarations, Definitions and Function Calls	43
8.1.2	Types	44
8.1.3	Logical Expressions	44
8.1.4	If Statement	44
8.1.5	While Loop	45
8.1.6	Count Loop	45
9	MissionEDU Client	46
9.1	Overview of Client Frameworks	46
9.1.1	WPF	46
9.1.2	Mono	46
9.1.3	Cocoa	46
9.1.4	QT	46
9.1.5	Cordova	47
9.1.6	Ionic	47
9.1.7	Xamarin	47

9.2	Xamarin for MissionEDU	47
9.3	MissionEDU Client Project Structure	48
9.4	MissionEDU Client Architecture	48
9.5	Network Layer	49
9.6	Task Loading	50
9.7	Command Loading	50
9.8	Program Functionality	50
9.8.1	Program Creation and Switching	50
9.8.2	Program Execution	50
9.9	Graphical User Interface	50
9.9.1	Start Page	51
9.9.2	Task Selection	51
9.9.3	Programming Page	51
9.10	Settings	52
9.11	Help Functionality	52
10	Conclusion	54
10.1	Applications of MissionEDU	54
10.1.1	K12 Student Courses	54
10.1.2	Alternative Programming Interface	54
10.1.3	Controller Environment and Front-End	54
10.1.4	Bridging a fragmented eco system	54
10.2	Outlook	55
10.2.1	Release	55
11	Acknowledgement	56
	Bibliography	57

Diplomarbeit Dokumentation

Namen der Verfasser/innen	Daniel M. Swoboda Markus Pinter
Jahrgang Schuljahr	5AHIF 2016 / 17
Thema der Diplomarbeit	An Open Approach to Educational Robotics Systems
Kooperationspartner	F-WuTS

Aufgabenstellung	Entwicklung eines Server-Client-Systems um beliebige Robotik Controller über eine Tablet-App und eine Netzwerkverbindung unter Zuhilfenahme einer graphischen Programmiersprache programmieren und steuern zu können.
------------------	---

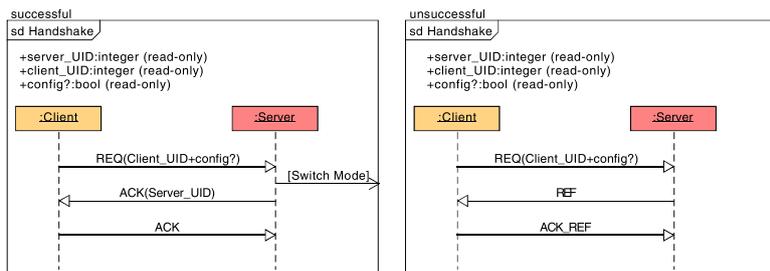
Realisierung	Realisierung mithilfe von Python sowie C++ auf der Server Seite, C# und Xamarin auf der Client Seite, sowie C++ und Qt für die Konfigurationsapplikation.
--------------	---

Ergebnisse	Funktionsfähiger Server Funktionsfähiger Programmier-Client Funktionale Programmierumgebung Funktionale Konfigurationsapplikation
------------	--

Typische Grafik, Foto
etc. (mit Erläuterung)

UML Sequenz Diagramm zur Darstellung des
MiRACLE Protokoll Handshakes

UML Sequence Diagram - MiRACLE Handshake



Teilnahme an
Wettbewerben,
Auszeichnungen

European Conference on Educational Robotics 2017
- Paper Submission

Möglichkeiten der
Einsichtnahme in die
Arbeit

HTBLuVA Wiener Neustadt
Dr.-Eckener-Gasse 2
A 2700 Wiener Neustadt

Approbation

Prüfer

Abteilungsvorstand

(Datum, Unterschrift)

MMag. Dr. Michael Stifter

AV DI Felix Schwab

Diploma Thesis Documentation

Authors	Daniel M Swoboda Markus Pinter
Form	5AHIF
Academic Year	2016 / 17
Topic	An Open Approach to Educational Robotics Systems
Co-operation partners	F-WuTS

Assignment of tasks	Development of a Client-Server-System to control robotic controllers over a tablet-app and network connection through an graphical programming language.
---------------------	--

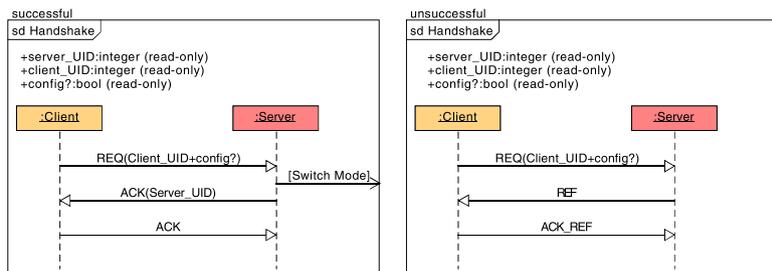
Realization	Realization of the server with Python and C++, the tablet client with C# and Xamarin, and the configuration app with C++ and Qt.
-------------	--

Results	Functional server Functional programming-client Functional programming-environment Functional configuration-application
---------	--

Illustrative graph, photo
(incl. explanation)

UML Sequence Diagram depicting a successful and unsuccessful handshake

UML Sequence Diagram - MiRACLE Handshake



Participation in
competitions,
Awards

2017 European Conference on Educational Robotics
- paper submission

Accessibility of diploma
thesis

HTBLuVA Wiener Neustadt
Dr.-Eckener-Gasse 2
A 2700 Wiener Neustadt

Approval

Examiner

Head of Department

(Date, Sign)

MMag. Dr. Michael Stifter

AV DI Felix Schwab

Kurzfassung

Diese Diplomarbeit stellt das bildungsorientierte Robotik-Toolkit MissionEDU, ein unter open-source Lizenzen veröffentlichtes System, das Unterstützung für graphische Programmierung (GP) für die meisten, modernen Robotik Controller nachliefern kann, dar.

Es besteht aus einem Server, welcher auf dem Robotik Controller läuft, einem Client für Tablet-PCs und einer Konfigurations-Applikation. Das System ist als umfassende Lösung für Robotik-Bildung gedacht und entspricht den Anforderungen eines Robotik-Kurs-Curriculums.

Zu den Fähigkeiten von MissionEDU zählen die Möglichkeit Programme für Roboter komplett in den grafischen Blöcken einer grafischen Programmiersprache (GPS) mit einfachen Kontrollstrukturen komplett auf einer Tablet-App zu schreiben, die Möglichkeit Funktionen die den Usern bereitgestellt werden hinzuzufügen und zu bearbeiten, die Möglichkeit sogenannte "Tasks" Schüler zu schreiben und via der App bereitzustellen sowie eine Methode der Abstraktion von Robotik Systemen um Plattformunabhängigkeit zu gewähren.

GP wird als eine der einfachsten Arten Programmieren zu lernen, besonders im Bereich der K12 (Kindergarten bis 12. Schulstufe) und für jüngere Programmieranfänger, angesehen und wird im Bildungsbereich seit mehr als einem Jahrzehnt vielfältig eingesetzt. Während einige moderne Robotik Systeme bereits GPS inkludieren gibt es kein System, das sowohl einen Fokus auf Bildung legt und gleichzeitig eine Programmierumgebung (PU) für eine große Zahl verschiedener Robotik Controller bereitstellt. MissionEDU versucht diese Lücke zu füllen.

Durch seine Möglichkeiten kann MissionEDU auch gegen das große Problem der Systemfragmentierung eingesetzt werden indem es eine universelle PU für eine heterogene Sammlung von Robotik-Controllern bereitstellt. Aus diesem Grund kann MissionEDU auch dazu beitragen die Ressourcen die einem Robotik-Kurs bereitstellen zu vergrößern indem es gewartete und aktuelle Software für eine Vielzahl von Controllern bereitstellt. Verschiedenste Controller können dank MissionEDU auf die gleiche Art und Weise programmiert werden, was den Zeitaufwand im Unterricht reduziert.

Abstract

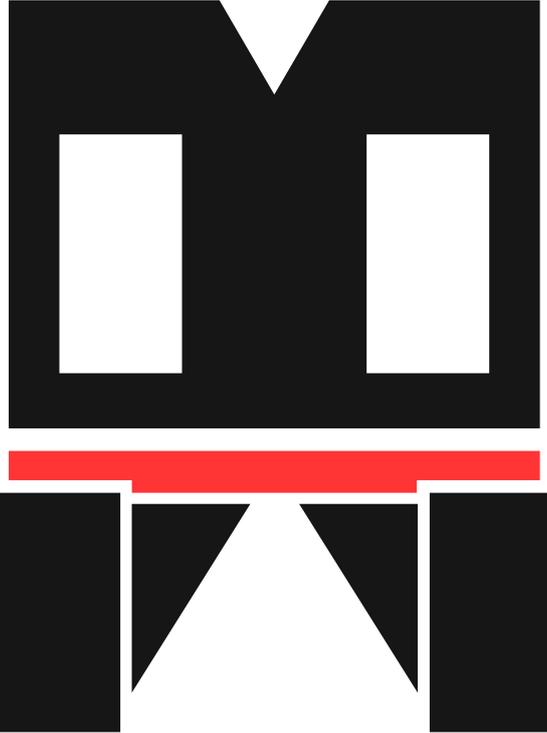
This diploma thesis introduces the educational robotics toolkit MissionEDU, an open-source system that can add support for graphical programming (GP) to most modern robotics controllers.

It consists of a server - running on a robotics controller, a client for tablet PCs, and a configuration app. The system is designed to be a self-contained solution for education meeting the needs of robotics course curricula.

It's features include the ability to program a robot completely in graphical blocks with simple flow control structures on an tablet PC, the possibility to add and edit the methods provided to the users through the configuration app, the ability to write tasks for students (with constraints to what blocks can be used) and provide them to the students via the tablet apps, and a method of abstraction of robotics systems in order to enable platform portability.

GP is considered the easiest way to learn computer programming especially for K12 students and younger programming beginners, being used in education for more than a decade. While some modern robotics systems include a graphical programming language, there is no system that focuses on education and - at the same time - provides the same programming interface for a greater variety of different controllers. MissionEDU is aimed to fill this gap.

Through its capabilities MissionEDU can also be used to counteract system fragmentation by providing a unified programming interface (PI) to a heterogenous set of robotics controllers. It therefore can increase the resources available for robotics courses by providing maintained and topical software for the controllers. Different controllers can be programmed in the same way using MissionEDU which decreases the amount of time necessary to introduce different systems.



MissionEDU

Chapter 1

Introduction

The economy of the future will be shaped by computers even more than it is nowadays. From the ever growing fields of robotics and software development, which are the key industries to require strong programming skills, to design, production and administration, computer science knowledge will be required more and more. Getting the youth into programming should therefore be one of the key aspects of our education and educational programs. To spark the interest of young people into the so called STEM subjects many educational robotics programs were established in the last two decades. [1]

These robotics programs often use the C programming language and some sort of Linux based robotics controller, thereby teaching kids and teenagers programming in one of the most important programming languages and Linux skills as well as knowledge about micro-controllers, mini PCs, mechanics and robot construction.

Besides of educational robotics programs a movement that's trying to get kids into coding with the help of so called Visual Programming Languages (VPLs) arose within the last decade when the educational VPL 'Scratch' was released. Programs and products like code.org, the Lego Mindstorms and Open Roberta make use of Visual Programming to get even the youngest into programming for whom text based programming languages are often hard to understand.

1.1 Task

Author: Daniel Swoboda

The task of this diploma thesis is to design and develop the server-client-system MissionEDU, that allows remotely executed programs to transfer commands in real time over an established computer network to a robot which then executes the received instructions. MissionEDU is meant to be used in robotic and computer science beginner classes, targeting K12 students. The remote programs are created using a graphical programming language specially designed for this purpose, called MateScript. It's simplicity makes it possible that programming beginners are able to use MissionEDU to learn how robots work and how to develop programs for them. In order to enable it to run on multiple robotic controllers, a way of creating an abstraction of a robotics system is necessary. The clients, which run on Android and iOS driven tablet PCs, receive a list of available commands from the robot they are connected to. These instructions can be created by the supporting teachers, the community and the users. They are meant to be customizable through an desktop application that allows to configure the robots.

1.1.1 Features of MissionEDU

MissionEDU consists of several features which are combined to form a singular experience for the users. The features are introduced and described in greater detail in Chapter 3 and following.

Features on the Robot Controller side are:

- Robot Definition Language
- Robot Abstraction Layer
- Robot-Server Interface
- Server
- swockets
- Networking Protocol
- Configuration Application

Features on the Tablet App side are:

- MateScript Programming Language
- MateScript Parsing Engine
- App User Interface
- Program Design Area
- Client

During the development the concept of visual programming languages and the way educational robotics programmes are designed were of great importance.

1.2 Visual Programming Languages

Author: Daniel Swoboda

1.2.1 The History of Visual Programming Languages

Ever since Ivan Sutherlands Sketchpad brought direct graphical communication between a human operator and a computer into reality there was a desire to integrate more natural ways of communication into the world of computer science [2]. With computers getting more and more powerful and the need for programmers rising Visual Programming Paradigms shifted from purely theoretical works to more and more advanced practical implementations. Evolving from the first visual dataflow languages of the late 1960s visual programming as we know it nowadays came into existence. The first icon-based programming environment was created as a part of David Canfield Smith's PhD dissertation "Pygmalion: A Creative Programming Environment". Pygmalion also made use of the concept of programming-by-example wherein the user shows the system how to perform a task in a specific case and the system uses this information to generate a program which performs the task in general cases [3] [4]. These visual programming languages were originally invented in order to empower end users of computerized systems, often office workers with no computer science background, to write simple programs to enhance their workflow and interact more deeply with the systems they were using. With the advancement of graphical user interfaces and basic knowledge of computers getting more widespread these early visions of the Visual Programming Language as a tool for anyone died off until a few projects emerged that used VPLs educational aspects to teach to concepts of programming. These education focused VPLs like Scratch are the biggest area of application for Visual Programming nowadays.

1.2.2 Defining Visual Programming Languages

A visual programming language is a programming language that uses visual presentation, such as graphics, drawings, icons and animations to communicate with a computerized system and program it [5]. Using an easy to understand icon based language to generate code that can be optimized and translated into machine code can be seen as the next evolutionary step from higher programming languages which brought programming closer to natural human language.

In order to classify Visual Programming Languages one has to consider the work of Chang, Shu and Burnett who defined the characteristics of the major categories of VPLs. The classifications can be summarized into 5 categories: Purely visual languages, hybrid text and visual systems, programming-by-example systems, constraint-oriented systems and form-based systems. [2]

The most relevant and commonly used concepts are purely visual languages and hybrid text and visual systems. While purely visual programming languages are only those where the user creates, arranges and changes icons or other graphical representations which are subsequently debugged and executed within the same environment to write a program, hybrid textual and visual languages use the same graphical elements but translate them into a high-level textual language, either an already existing one or a language created for the environment, which then is compiled or interpreted and executed [6] [7] [8]. Purely visual languages can be further subdivided into sections like iconic and non-iconic. The general classification of programming languages which include categories like object-oriented, functional, and imperative languages can be applied to all visual languages.

Examples of purely visual languages include VIPR, Prograph, PICT and the educational language Scratch while hybrid systems include Rehearsal World [2].

1.2.3 Specifying Visual Programming Languages

In order to formally specify the language design itself one has to distinguish three construction rules, used to arrange icons: horizontal linking, vertical linking and spatial overlay.

It is customary to distinguish several different types of icons, with the two main categories being process icons and object icons. Object icons can be further subdivided into elementary object icons and composite object icons. The former represent an primitive object of the language, like the data type integer does in higher programming languages. The latter represent complex objects that can consist of several elementary and complex objects, they most closely resemble structs. Composite objects can also be created by spatial overlay of elementary object icons. Process icons on the other hand describe operations performed on object icons, these processes can be anything from mathematical operations to highly complex algorithms. [2]

1.2.4 An overview of Educational Visual Programming Languages

Educational Visual Programming tries to use the ideas and concepts of VPLs in order to make it both easier for kids to enter programming and make it more appealing than plain text. One of the first applications of Visual Programming Languages in educational programming was the use of them for the Lego RCX, the first controller of the 'Mindstorm' Series of products. Beginning with Scratch, an educational VPL developed by members of the MIT Media Lab in 2007, these educational VPLs became more and more widespread and separated from robotics. Since then other platforms like code.org and Open Roberta were established and created additional possibilities to learn programming with visual concepts.

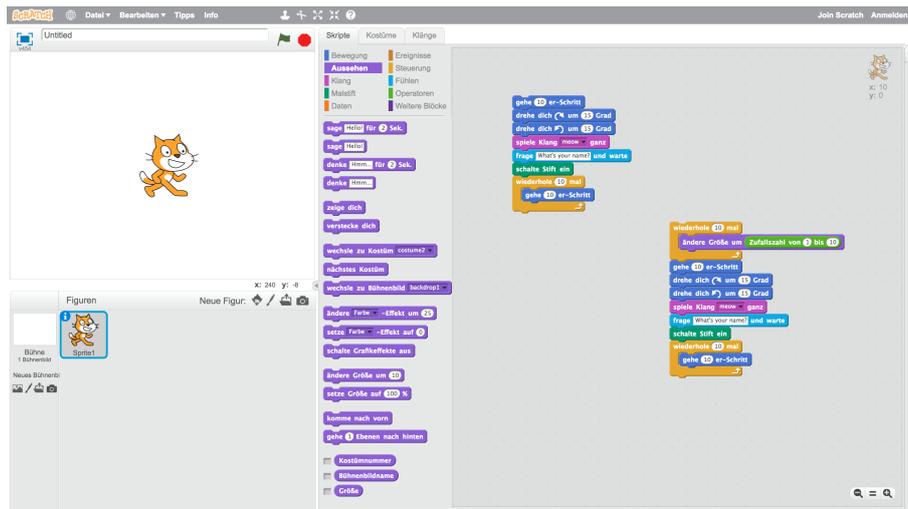


Figure 1.1: The Visual Programming Language of Scratch in the official editor

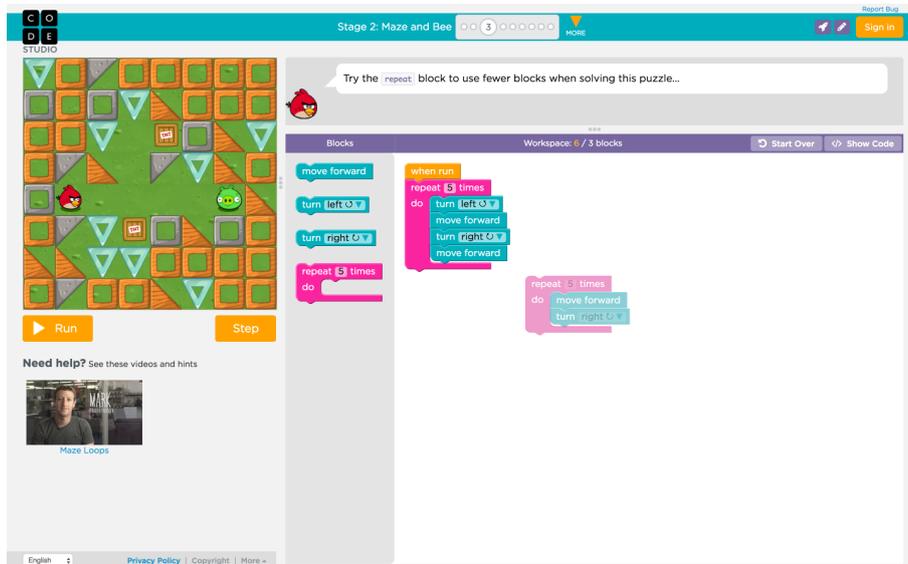


Figure 1.2: The Visual Programming Language of Code.org in the official editor

Lego Mindstorms is a series of products by the danish toy company Lego as part of it's pursuits in the educational sector. The first product of the Mindstorms series was the RCX in 1998 followed by the NXT in 2006 and recently the EV3 in 2013. All of the above have in common that they are designed as central 'bricks' to build a robot around. They also have a similar layout with a monochrome display in the middle of the brick and the ability to control 3-4 servos and motors while getting input from 3-4 sensors. All products of the series can be programmed in C dialects and an accompanying graphical programming language. This allows the Mindstorms robots to be used by absolute beginners as well as advanced users. Because of that and their relatively low price they are used in educational robotics programs around the world.

The Visual Programming Language Scratch was first introduced in 2003 with a subsequent public release in 2007. The key goal of Scratch is to introduce programming to those with no previous programming experience. It is one of the most widely used educational

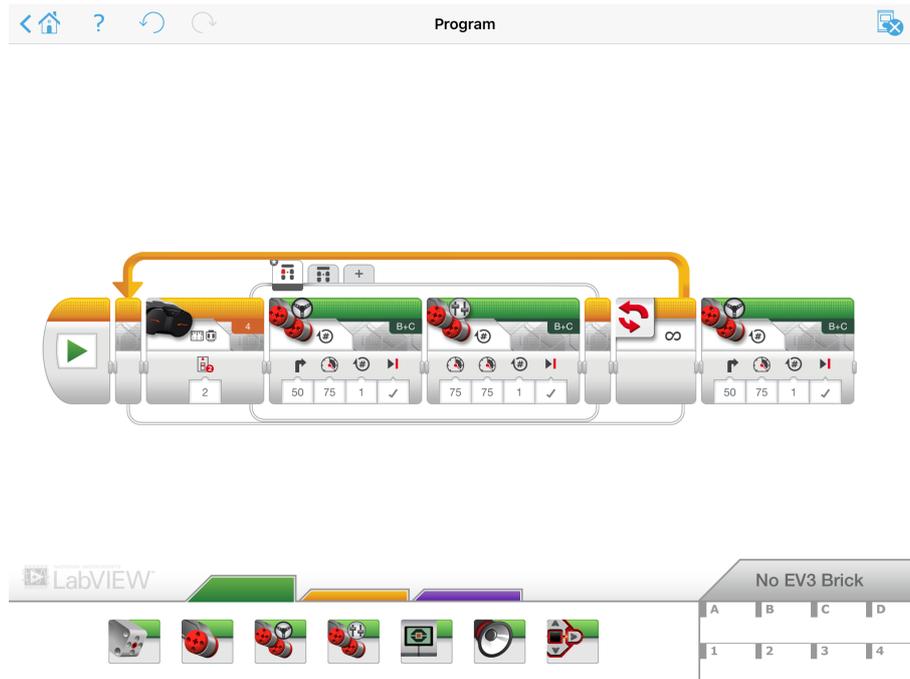


Figure 1.3: The Visual Programming Language of the Lego EV3 in the official iOS App

tools in the world with more than 2 million downloaded copies from the official website. Scratch tries to make programming appealing to the K12-students who have never programmed before by focusing on manipulating media content and using self-aligning colored blocks that snip together [9].

Code.org uses an Scratch-like Visual Programming Environment integrated into an online platform. It was launched in February 2013 and is supported by public figures like Bill Gates, Jack Dorsey, Mark Zuckerberg and Bill Clinton. The platform depends on partner companies to develop so-called playgrounds, which are the tasks the students have to solve [10].

Especially for K12-students, visual programming can greatly help with lowering the barrier of entry for educational robotics programs.

1.3 The State of Educational Robotics

Author: Markus Pinter

During the last 40 years, robotics plays an important role in the industry. Autonomous systems became essential, thus informatics gained high demand in this industry. Educational Robotics encourages young people dealing with new technologies. This can be achieved by making programs, where students learn about building and programming robots in a funny and inspiring way.

There are many Programs about Educational Robotics all around the globe, commonly organized as competitions, where students meet up, share their experiences they made and in the case of an affiliated conference listen to talks. In the United States, the Global Conference on Educational Robotics (GCER) and in Europe the European Conference on Educational Robotics (ECER) are held every year [11] [12]. During the conference, students can improve their knowledge of how robots interact with the physical environment as well

as getting a comprehensive introduction in the field of robotics.

The talks are usually held by people who are working at a commercial or research driven robotics institute. They give students a picture of how such projects look like in industry, by telling them the tasks they had to achieve. Furthermore, they speak about certain challenges during their work and how they coped with them. At the end, there's a thorough Q&A session for students to ask questions they might have.

1.3.1 Competitions

Competitions are very important in the field of educational robotics. Some of them allow only specific hardware parts whereas others don't have any restrictions. Competitions often consist of more than one discipline where the focus of each discipline differs content-related.

The affiliated robotics program to GCER is Botball, where the tasks of the upcoming event are usually presented in January [13]. Both, the rules and the equipment are defined. The tasks have to be realized on a specific game table, where the robots start autonomously for a period of two minutes. Afterwards, the robots must stop autonomously as well and during the 2 minutes, it is not allowed to interact with the robots.

Botball takes place alongside GCER and is divided into four disciplines. The first one is all about documentation, where you need to write down problems that occurred as well as experiences gained during the preparation phase. In the second discipline, so called Seeding, each team starts alone on one side of the table and may also score points on the other side. After three runs the team with the highest average of its best two runs, wins the discipline. The third called Double Elimination is about two teams competing against each other on the same game table. The winner is whoever has more points at the end of the run. This discipline is structured in brackets which I won't go into detail any further due to complexity. Every team that hasn't made it to the finals of Double Elimination gets another chance in the so-called Alliances. Two teams need to work together and now may use two kits instead of one. It's just like Seeding with the only exception that each team starts on one side of the table and scored points are added up at the end of the run.

Another well-known competition within this topic is The Robot World Cup Initiative (RoboCup). At the seniors level, the main goal is to be able to play soccer with completely automated robots against human World Cup winners until 2050. Therewith they want to push robotics research, such as design principles of autonomous agents, multi-agent collaboration, strategy acquisition, real-time reasoning, and sensor-fusion [14]. Currently five major domains exist.

- RoboCupJunior Soccer
- RoboCupJunior Rescue
- RoboCupJunior OnStage

The first one is RoboCupJunior Soccer, where two teams with up to 2 robots have to play soccer against each other. This is a very ambitious task because one has to take many physical parameters into account to react to all different situations that might occur during a game. Therefore very expensive hardware is used to achieve a high accuracy in moving the robot as well as detecting surroundings in the environment.

RoboCupJunior Rescue focuses on finding an objective or victim and transporting it to a safe zone. The environment might include lines to follow, gaps between them and crossings to overcome.

In RoboCupJunior OnStage the best robot dance performance with the participants win.

1.3.2 Junior Competitions

Although a great variety of competitions already exist for young people, they aren't really developed for kids younger than 10. Therefore distinct challenges were invented that are easier to accomplish to avoid frustration if the complexity, e.g., in the syntax of a programming language is too high. The definition of whether one's junior or not varies between competitions.

For pupils in the age of 6 to 12, tasks might look like just following a line, or in addition grabbing an object. The goal is that students should try different approaches to solving different tasks. These tasks emphasize the interaction between the robot and the physical environment. To program their robots they mostly use visual programming languages, such as the LEGO Mindstorms Programming Software.

Students from 13 to up to 19 years typically have more advanced tasks to fulfill. For example driving up a ramp, driving around obstacles that are in the way and transporting objects from A to B. In some competitions, positions of objects may be random and change from run to run. Especially older students tend to use text-based programming languages such as C or NXC because they can write programs in a more flexible manner.

An important thing to note is that hardware costs of such junior challenges are much lower than for regular ones. This is associated with the fact that most schools cannot afford such expensive hardware as universities can do.

1.3.3 Hardware used in Educational Robotics

Since building hardware controllers can be a very complex task, companies started projects to develop controllers specifically made for students. The probably most popular series is called Mindstorms from LEGO. These controllers are also called bricks because they can easily be attached to other Lego parts. The most modern model is called EV3 and has a much higher processing power and memory than its ancestor NXT.

Arduino boards are also very popular for usage as robot controllers, since they are very cheap and can be used for various tasks.

1.4 Enabling Robotics Education through Visual Programming

Author: Daniel Swoboda

Since programming techniques in educational robotics programs tend to be similar there is a great potential for standardization and the use of one tool for multiple platforms and contests. Linear program structures that are easy to comprehend, learn and design, especially by young students, are perfect to be represented by an icon-based VPL. Since young students often show a natural interest in robotics and moving parts in general and their familiarity with the use of Tablet PCs and mobile apps the combination of both comes naturally.

MissionEDU is intended to close the gaps between different platforms and to provide a singular experience across a great variety of different controllers. With a simple user interface and an easy to understand graphical user interface a system like it enables more students to get access to educational robotics. It allows students of younger age to gather experience on different platforms. MissionEDU also aims to help institutions with lower budgets to incorporate a fragmented hardware eco-system by unifying the programming interface.

Chapter 2

Project Management

Author: Daniel Swoboda

Every larger project - especially ones with multiple persons involved - need adequate project management in order to be carried out successfully [15]. Since MissionEDU is a diploma project and therefore had to be developed outside of school times a flexible and agile kind of project management that emphasizes task achievement and doesn't require static milestones or daily meet-ups was needed. Because of this the decision was made to use Kanban.

2.1 Kanban

2.1.1 Description of Kanban

Kanban is a popular and simple technique for managing tasks in an ordered and effective way. Its name derives from the japanese word for "sign-card". It enables teams to visualize the process and by that decreases the chance for bottle-necks to occur. Rather than focusing on reaching milestones, it focuses on achieving the tasks necessary without a given time-frame for each task. [16]

Its main principle is the "Kanban board" which uses "cards" to create an visualization of tasks that are yet to do, in progress and already done.

2.1.2 History of Kanban

Developed by Toyota in the 1940s to optimize workflow and introduce Just-in-Time production, it is based on a method supermarkets used to restock. The goal was to better align the massive inventory levels of their production facilities to match the actual need. To communicate the capacity levels "Kanbans" where used. [17]

Agile software development teams took the basic principles and applied them to software development. The parts inventory was replaced with the tasks that are in the three different states and the board is used to signal how many are in each state. [17]

2.1.3 Application of Kanban in this project

Kanban was the main method of project management applied during the work on this thesis. There were 5 different Kanban boards in use with one person assigned per board:

- Server Kanban board (Swoboda)
- Configurator-App Kanban board (Swoboda)

- Client-App Kanban board (Pinter)
- Graphical Programming Language Kanban board (Pinter)
- Abstract Robotics System Kanban board (Swoboda)

2.2 Results

The project was completed in time and according to the specifications submitted project description. However, in agreement with the project advisor the early schedule was moved behind by 1 month and the late schedule by 2 weeks to better integrate it into the expected schedule.

2.2.1 Schedule and content of meet-ups with the project advisor

Meetings with the project advisor were held every week on friday. In these meetings the project advisor was informed about the current state of the project and could be asked questions related to the development or writing process.

2.2.2 Hours worked

Over the course of the project the authors worked on their parts approximately 180 hours each. The time can be split into hours working on the project itself and hours writing this thesis as depicted in Table 2.1).

Table 2.1: Hours worked per person and topic

Name	Hours coding and project management	Hours writing
Markus Pinter	100 hours	80 hours
Daniel Swoboda	90 hours	90 hours

Chapter 3

Networking Technologies

Author: Daniel Swoboda

3.1 Networking in MissionEDU

As MissionEDU is a client-server-system, a common standard for communication had to be created. One important part of the project are networking technologies. Modern robotics controllers, especially the ones used in educational programmes, are similar to smartphones in a lot of aspects [18] [19]. Hence, they support standard technologies like Ethernet or Wireless-LAN to connect to a Local Area Network (LAN). Since Wi-Fi is one of the most used computer networking methods, being already available in most schools and homes, the decision was made to focus on Wi-Fi as the main medium for network access [20]. Computer networks using these technologies are cheap and easy to set-up [20]. Most controllers already require a W-LAN for operation. Wireless LAN is also available in most Tablet-PCs, MissionEDU therefore uses TCP/IP over LAN to establish a connection and exchange data between the server and the clients. The controllers and Tablet-PCs supported by MissionEDU have to provide a socket interface and a way of connecting to the network.

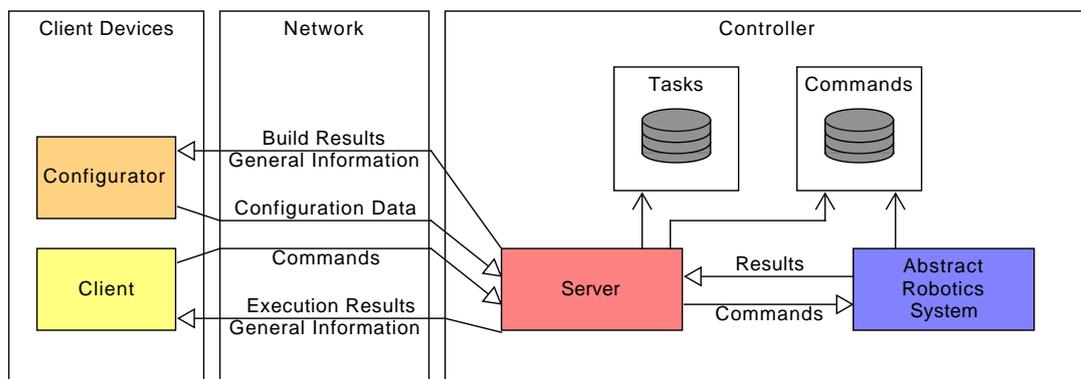


Figure 3.1: The data flow between the elements of MissionEDU

3.1.1 Data Flow

Data is exchanged within a MissionEDU setup bidirectional between the server and the connected clients. The available commands are sent to the clients on connection. Sensor data can also be requested from the server whenever needed by an executing program. Configuration clients send updated tasks and commands to the server while programming clients send instructions for execution. A representation of the data flow within a MissionEDU setup is depicted in Figure 3.1

There is no need for communication between the connected clients, therefore it isn't supported, neither by the server nor by the protocol. The configuration client is able to connect to multiple clients at once when updating configurations but each connection is handled as a distinct and isolated structure. It is not intended that multiple servers communicate with each other.

3.1.2 Data Encapsulation - Data Interchange Format

A common format for messages and data serialization is needed for every network based application. In MissionEDU the JavaScript Object Notation (JSON) standard is used. Messages formatted in JSON are abstract ASCII representations of objects with attributes, arrays and values (example depicted in Code 3.1).

Code 3.1: Example of JSON formatted data

```
1 {  
2   "Name" : "Bob",  
3   "Address" : ["Main Street", 10, 1080, "Vienna"],  
4   "Age" : "10"  
5 }
```

JSON is supported by all common programming languages either as a part of the standard libraries or through third-party ones. This makes it ideal for a setup like that of MissionEDU where multiple programming languages on multiple platforms are used. It is designed to be both: readable by humans to reduce debugging efforts and easy to parse by computers. [21]

Compared to XML - which the most common standard for data interchange - JSON has less overhead and therefore better performance. The smaller message sizes also result in an increase of the amount of messages that can be sent per time unit. [21]

Binary data can be converted into the Base64 format and included within the messages as a string. While this leads to an increase of the amount of data by 30% and therefore isn't the most efficient way of transmitting binary data, it is more suited to the needs of MissionEDU than other methods especially since the transmission of binary data isn't expected. [22]

3.2 MiRACLE Protocol

In order to have a controlled flow of data between the components over the network an application layer protocol has to be used. A protocol also pre-defines message layout and specifies the behavior for each event. It also includes a set of messages known to all communicating parties that acts like a common language. [23]

The MissionEDU Robot Access, Configuration Loading and Enhancing Protocol (MiRACLE) is the protocol used for all communication between the different clients and the server. It is designed to work on a TCP connection, but could theoretically also be used

with any other transport layer protocol that provides reliable and error-checked delivery of data.

3.2.1 Defining Protocols

"Protocols are sets of rules that govern the interaction of concurrent processes in distributed systems" [23].

Communication protocols existed ever since humans developed information transmission systems that worked over greater distances. One of the first examples of communication that required a protocol was a system of beacons on towers used by the ancient greeks. This system enable the transmission of results from battles over great distances in comparatively short time. The 'protocol' here had to describe what a lit fire on another beacon meant to the observer and what the operators had to do when they spotted light on the next station. Other classic forms of communication that required protocols to be understood and used were the french optical telegraphs of the 18th century, which delivered messages to almost all ends of France. With the dawn of the electrical telegraphs and the telephone standardized protocols gained even more importance. [23]

The first protocols for inter-computer communication were the master-slave protocols used to control the peripheral devices of mainframe computers like the ENIAC. Modern communication protocols, like TCP/IP, were first developed by researches to be used in the ARPA-Net. [23]

Because of the prevalent use of distributed communicating systems a well designed and valid protocol is important. In software development bad protocol designs can lead to inefficient programs or worse, they can lead to software crashes and incoherent states. [23]

3.2.2 Protocol Specification Theory

Within a precise specification all aspects of the protocol have to be gathered and described. Such a specification consists - according to Gerard Holzmann - of five elements. These rules make sure that each included element of the protocol is described and coherent [24]:

- The service to be provided by the protocol
- The assumptions about the environment in which the protocol is executed
- The vocabulary of messages used to implement the protocol
- The encoding (format) of each message in the vocabulary
- The procedure rules guarding the consistency of message exchanges

Working with this ruleset reduces the amount of mistakes that are likely to be made by inexperienced protocol designers. While developing the MiRACLE protocol these rules were applied and the design was tested and adapted in an iterative process. Describing the service first and making assumptions about the environment early-on made it easier to create a flow model of the protocol. With the flow model, both under-specifications and over-specifications, are easier to detect, which led to a more consistent and efficient design.

3.2.3 Protocol Design Theory

In order to help with protocol design - which is based on the previously created specification - a set of design rules can be applied. These rules are based on the five elements of a protocol specification by Holzmann and adapted for usage in protocol design [25]:

- To make sure a protocol is optimally designed for it's job the problem has to be well designed and the service has to be described first.

- External interfaces and functionality have to be designed before one designs internal functionality in order to keep the interface simple.
- Independent parts, like protocol interfaces and the implementation of interface access, have to stay separate.
- In order to guarantee extendability as few restrictions as possible should be imposed.

For the design and development of a protocol a variety of strategies known from project management theory can be applied. Applicable strategies include the build and-fix-model, the waterfall model, the V-model, evolutionary development and Boehm's Spiral Model [26]. The model used to develop MiRACLE was evolutionary development since it is easy to apply and integrate. This technique also produces many intermediate versions that can be tested to find the best working design.

Designing a protocol requires a formal way of describing it's workflow and interfaces. Graphical models are especially suitable for this task [27]. The diagram-types most widely used are:

- State-Transition-Diagrams, to describe a protocol from the point of view of valid program states
- Sequence Diagrams, to describe the chronological flow of a protocol
- Finite State Machines, as an abstract representation of program flow

For the description of MiRACLE an UML Superstructure Specification Diagram was used to create a State-Transition-Diagram and a UML Sequence Diagram was used to describe specific parts of the protocol flow.

3.2.4 Transition and Sequence Specification of MiRACLE

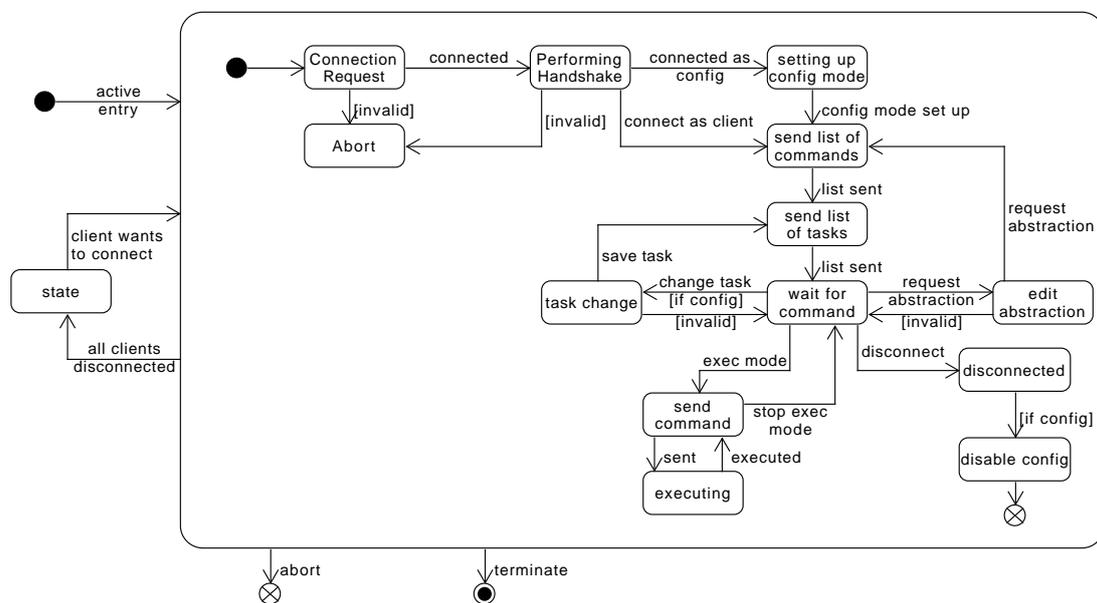


Figure 3.2: UML superstructure specification diagram as a state-transition-diagram of the MiRACLE protocol

A state-transition diagram of the MiRACLE protocol is depicted in Figure 3.2. It which describes the transitions between the different states of the protocol and the events which

lead to their occurrence. It also describes how the state of the server when no client is connected and how origin and final states are entered.

On the establishment of a connection to the server the client performs the handshake and negotiates the conditions of the connection with the server. After a successful handshake a standard client receives a list of the saved student tasks and a list of all the commands available. A configuration client receives the same lists, but in addition the server switches to configuration mode. In it MissionEDU-Server performs a disconnect on all other clients and a shutdown of the abstract robotics system, since it is expected that it will be altered and recompiled. After sending the tasks and commands to the clients, the server switches to an idle state, waiting for incoming commands. What the diagram doesn't show is the multi-threaded nature of the protocol meaning that a server implementing the protocol is able to accept multiple connections at once. This is however limited in implementation because of the low specs of the target devices.

In the idle state configuration clients can...:

- change and update the commands
- change, update and add the list of tasks

...standard clients can...:

- start the execution mode
- send commands to control the robot, when in the execution mode

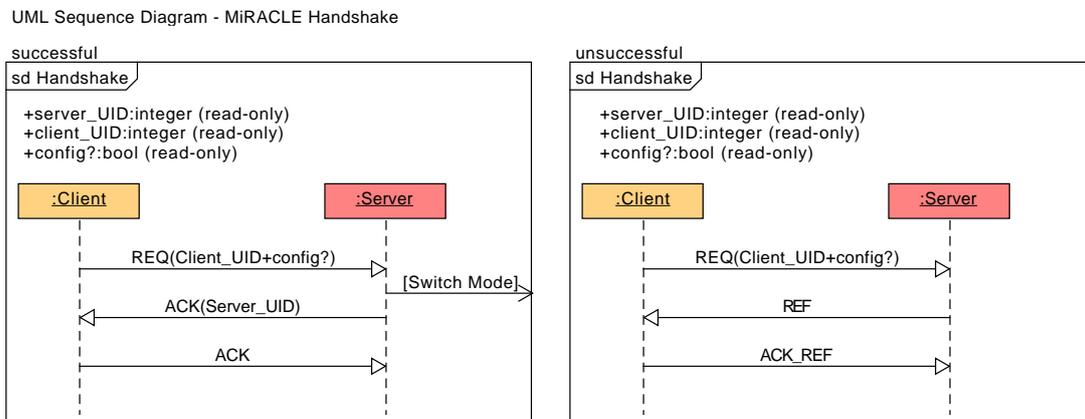


Figure 3.3: UML Sequence Diagram depicting a successful and unsuccessful handshake

Figure 3.3 depicts the chronological flow of the handshake of the MiRACLE protocol. A connecting client transmits its mode (standard or configuration mode) and its UID generated on installation. If the server is able to accept new connections it will answer the request with an acknowledgement message containing its UID. The client confirms the acknowledgement and the protocol transitions to the next state.

If the server can't accept new connections it sends a refusal message (REF). The client has to accept it with a refusal acknowledgement (ACKREF) and the connection is closed.

3.2.5 Data Format and Message Specification of MiRACLE

Structuring protocol messages is an important task since a common data format and a specification thereof is required to assure interoperability of the various components. As JSON was selected to be the only data format that is used for messages within MissionEDU, MiRACLE is designed to solely use JSON formatted messages. Because of the common format between all components no intermediate data-structure is necessary and no conversions have to be done. MiRACLE has a general message structure that is consistent across all messages. As it can be seen in Code 3.2, a message object starts with a message type description. A pre-defined message type must be chosen as the value. The data of the message is wrapped in the payload object, which has a pre-defined layout for each known message type.

Code 3.2: General structure of a MiRACLE message

```

1 {
2   "MessageType": "<type>",
3   "Payload" : {
4     <payload data for message type>
5   }
6 }
```

For non-data messages like ACK, which is used in the handshake, the payload is optional and can be left-out to save bandwidth. White spaces, line breaks and carriage returns can also be removed in order to further reduce the size of the messages.

3.2.6 MiRACLE Validation Model

Validation models for protocols are necessary to perform checks on the validity of the conceptualized protocol. Such a model also helps with creating a description of the procedure rules, a finite state model and a prototype of the implementation. To perform model checking - which is a task that allows for automated verification of the protocol and analysis of the logical properties of the protocol - the description language PROMELA (an example can be seen in 3.3) (Protocol / Process Meta Language) can be used. Combined with the Simple Promela Interpreter (SPIN) it allows for fast implementation and testing of network protocols and other reactive systems. [28]

Code 3.3: Example of a simple PROMELA model

```

1 mtype = { msg0, msg1, ack0, ack1 };
2 chan to_sender = [2] of { mtype };
3 chan to_receiver = [2] of { mtype };
4 proctype Sender()
5 {
6   again:
7     to_receiver!msg1;
8     to_sender?ack1;
9     to_receiver!msg0;
10    to_sender?ack0;
11    goto again
12 }
13 proctype Receiver()
14 {
15   again:
16     to_receiver?msg1;
17     to_sender!ack1;
18     to_receiver?msg0;
19     to_sender!ack0;
```

```

20 goto again
21 }
22 init{ run Sender(); run Receiver(); }

```

Using PROMELA and SPIN to create a validation model decreased testing time and helped with the elimination of logical errors in the protocol from an early stage on.

3.3 Socket Wrapper Library swockets

To reduce the effort needed in order to implement the MiRACLE protocol on all platforms and make it easier to work with communication end-points (called sockets) a library was needed that is available on all platforms. None of the existing libraries fullfills the needs of MissionEDU, so an socket abstraction library particular for the project - called swockets - had to be created. Swockets takes care of the TCP connection, format checking and sequencing of the messages, error handling and asynchronous I/O operations. The interface of swockets is designed to be as similar as possible across all platforms and programming languages. It provides both a server and a client mode and works event based. It supports the usage of self-defined event handlers and is designed with entry points for user code to allow for easy implementation of the desired protocol.

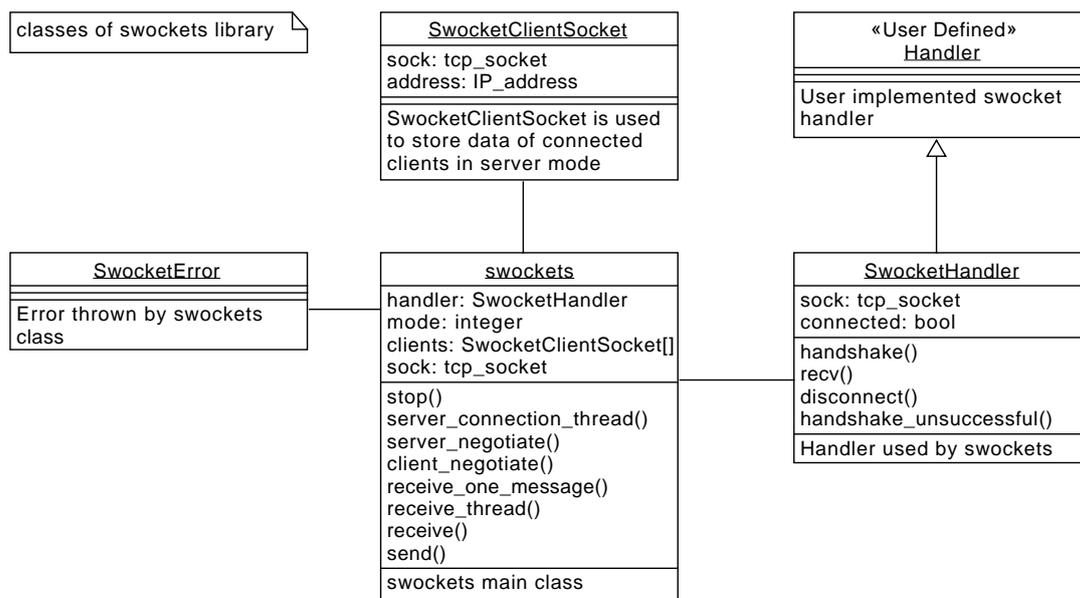


Figure 3.4: UML class diagram showing the class structure of swockets. The same structure is used in the implementation for every language and platform.

3.3.1 Structure

The structure of swockets is designed for object-oriented programming since all of the languages that need to access the network in MissionEDU can benefit from such a design. The class relations are depicted in Figure 3.4 and consists of the following parts:

- **BasicHandler** class: General implementation of the swocket event handler. Implements all functions called by swockets to communicate with the rest of the program.

An object of this type or of an inherited type is required for the construction of an `swockets` object.

- `swocket Handler`: User written handler, inherits from the `BasicHandler` class. Includes the handler implementation specifically needed for the program the `swocket` object is used in.
- `Receive Thread`: The receive thread takes care of incoming messages and calls the respective `swocket` handler method.
- `Send Method`: Used to send a message to a client (in server mode) or the server (in client mode).
- `Constructor`: Creates the `swockets` object in either server mode or client mode. Creates the TCP/IP socket, establishes a connection (in client mode), starts the TCP listener (in server mode), starts the receive thread.
- `Connector and Handshaker`: Called on `connect` (in client mode) or whenever a client wants to connect (in server mode). Calls the `swocket` handler `handshake` function, where the user specified handshaking mechanism is defined.
- `Client Handler (server mode only)`: Handles the connecting clients, performs the handshake specified by the user, automatically creates all objects necessary. Calls the respective `swocket` handler method.

3.3.2 Interface

`swockets` is designed to be easily adaptable to the users needs through a specialized event handling system and the `SwocketHandler` class. The customization process works by overloading the handler class. A object of this class is necessary to initialize a `swocket`. The functions in it are called on specific events during the execution of the program.

As a TCP socket abstraction, `swockets` main objective is to decrease the complexity of network communication development. Since it takes care of all critical aspects of networking it reduces the amount of critical code that has to be written and removes the needs of creating a threaded client handling system. This reduces the efforts necessary to create an application with network communication abilities. It also helps with generalizing the components of `MissionEDU` for multiple platforms because the platform specific code to access the native socket API otherwise be required to be written separately for each part of the project.

A `swockets` server or client is created in one line using the code depicted in Code 3.6 and Code 3.5. To implement your own functionality the `SwocketHandler` is overwritten (depicted in 3.4).

Code 3.4: Example of a partially overwritten `SwocketHandler`

```
1 class BasicHandler(SwocketHandler):
2     def __init__(self):
3         SwocketHandler.__init__(self)
4         self.connected = True
5
6     def disconnect(self):
7         self.connected = False
8         print "Server disconnected"
9
10    def handshake_unsuccessful(self):
11        self.connected = False
12        print "Handshake unsuccessful"
```

Code 3.5: Example of the initialization of a swockets object in client mode

```
1 handler = BasicHandler()
2 client = swockets(swockets.ISCLIENT, handler, 'localhost')
```

Code 3.6: Example of the initialization of a swockets object in server mode

```
1 handle = SsocketHandler()
2 server = swockets(swockets.ISSERVER, handle)
```

3.3.3 Program Flow

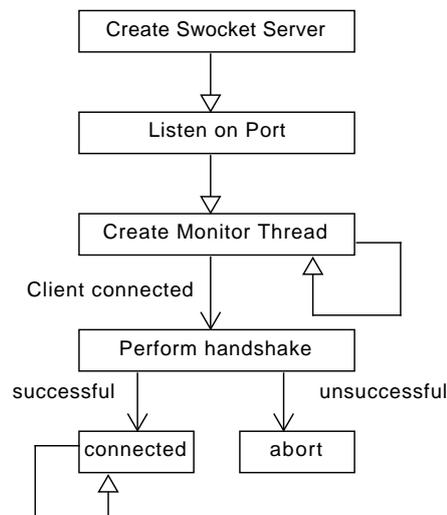


Figure 3.5: Program flow of a swockets server

As depicted in Figure 3.5 creating an swockets object in server mode, creates a TCP/IP listener socket on the specified port which listens on all networking interfaces of the device. A thread is started that monitors incoming connection requests created by clients. For every client that requests a connection a thread is started that is dedicated to handle the incoming messages of it. This increases the level of parallelization and decreases idle CPU usage by preventing the need for polling, which can be a CPU intensive task. Swockets then performs the handshake implemented in the handler that was passes as a parameter. If it is successful the client state is set to connected and it can now communicate with the server application. Otherwise it is disconnected and the user is notified through the call of an event. After establishing a connection the user - by default - receives all messages sent from the client through the "recv" event of the used handler. Optionally parallelization can be turned off if the application requires it.

If a swockets object is created in client mode it automatically establishes a connection through a TCP/IP socket to the given IP address and port that corresponds with the server. The handshake defined by the user in swockets handler is executed if the TCP connection was successful, otherwise it will be aborted. If the handshake was successful the swocket client can be used by the user in a similar way to the server, else the user is notified.

3.3.4 Implementation of MiRACLE with swockets

Since swockets is used as the socket abstraction, the MiRACLE protocol is implemented using the function calls provided by the library. A specific swocket event handler was created to handle the events and perform corresponding actions as specified by the protocol.

When a TCP connection is successfully established the handshake method is called on both the server and the client. The messages are exchanged as depicted in Figure 3.3

If the handshake is successful the connect function is executed. If the connecting client is of type configurator, the configuration mode is activated. In that case all other clients are disconnected and new connections are prevented. Within the connect method the client waits for two messages specifying the available commands and available tasks which are sent by the server. Subsequently the receive threads are launched on both sides, in which the incoming messages are handled. The program flow now is able to continue with the established connection and messages can now be exchanged.

Chapter 4

Abstract Robotics Systems

Author: Daniel Swoboda

There is a multitude of robotics controllers, programmable robots and micro controllers as well as micro PCs suitable for usage in educational robotics programs. These hardware systems (hereinafter 'controllers') use vastly different designs, libraries for programming and programming languages. (compare [18] [19])

MissionEDU is intended to run on as many controllers as possible. The goal is to enable this having to reimplement the server system for each controller. In order to accomplish this, an robotics system abstraction had to be created, which provides a uniform interface of access regardless of the underlying robotics system, operating system or hardware.

4.1 A Description of Modern Educational Robotics Controllers

A usual setup, common in educational robotics, consists of (compare [18] [19]):

- the controller hardware
- compatible sensors and actuators
- the operating system running on the controller
- the programming library needed for hardware access
- the robotics system which is responsible for managing the access
- additionally a tool to program the robot

The components vary greatly between the different eco-systems, are often incompatible and code can hardly be reused across different platforms. In the following sections example setups that can be found commonly are given.

4.1.1 Raspberry Pi Setup

The Raspberry Pi (hereinafter 'RasPi') is a cheap and powerful mini-PC (depicted in Figure 4.1, commonly used in educational robotics and school projects [30]. Being supported by a great variety of Linux distributions and Windows 10 makes it a versatile platform [31]. In order to get sensor data or control actuators can either be paired with a 'shield' which is an extension board for the RasPi, communicating with it over General-Purpose-Input-Output pins or the GPIO pins directly (which limits the Pi to only one hardware PWM channel) [32]. Shields are available from different suppliers, often as open source hardware. A RasPi shield usually comes with a Python or C++ library for programming and doesn't need a special programming tool. This means that the platform can be easily integrated



Figure 4.1: A Raspberry Pi 3 Model B [29]

into existing workflows.

Its specifications in Version 3 Model B are [33]:

- A 1.2GHz 64-bit quad-core ARMv8 CPU
- 802.11n Wireless LAN
- Bluetooth 4.1
- BLE
- 1GB RAM
- 4 USB ports
- 40 GPIO pins
- Ethernet port

4.1.2 KIPR Wallaby Setup

Note: Because of the lacking documentation of Wallaby hardware and software all information was gathered through tests on the hardware itself and information from the operating system.

The KIPR Wallaby is a robotics controller specially designed for the Botball competition by the KISS Institute for Practical Robotics (KIPR). Botball - with more than 180 participating teams each year - has its equipment used by almost 3000 students world-wide [34]. The Wallaby is similar to the Raspberry Pi yet less powerful when it comes to hardware specifications. It therefore comes with an integrated display and 12 ports for sensors (5 analog, 7 digital). The controller can also access up to 4 motors and 4 servo motors at

the same time. On the software side it runs a custom operating system which is based on 'Poky' by Yocto. Wallaby is produced by Gumstix via an on-demand production method. Programs are written in C++, using the built-in library. The intended way to program it is over an Web App running in a browser, hosted on the Wallaby itself. It isn't extendable by default.

Its specifications are:

- A 750MHz 32-bit single-core ARMv7 CPU
- 802.11n Wireless LAN in hotspot mode
- 512MG RAM
- 2 USB ports
- 12 sensor ports
- 4 servo ports
- 4 motor ports
- 1 micro USB port for cable based connection

4.1.3 Difference between Raspberry Pi and KIPR Wallaby

These two systems highlight the fragmentation of controllers that exists even if they have similar technical specifications. Code can't be shared between these platforms, because of the different operating systems and the lack of universal libraries. Binaries are incompatible.

Hardware can be shared because of the use of standardized ports for the actuators on the Wallaby and the RasPi shields.

An abstract robotics system therefore would make it possible to control the same set of hardware with the same code through different controllers. Old controllers could be easily replaced by new ones, with the same programs being able to run on them.

4.2 MissionEDU Abstract Robotics System

The Abstract Robotics System (or ARS) is the MissionEDU approach to solve this problem. A middleware running on the controller, provides a uniform interface to gain access to the robotics system and the underlying hardware. Its a distinct layer of the server stack running as an independent process on the controller itself. The ARS communicates over the standard Unix inter-process-communication method UDS (Unix Domain Sockets) with the server to allow for fast data interchange. It's design is inspired by the Hardware Abstraction Layer of modern operating systems.

4.2.1 Hardware Abstraction Layers

Diverse sets of complex systems long have been an issue in computer science, especially for operating system developers who are confronted with different instruction sets, platforms, graphic cards or even more specialized hardware designs. To create a universal way of accessing the underlying hardware, a so called Hardware Abstraction Layer (HAL) is used.

"The HAL is defined as all the software that is directly dependent on the underlying hardware." [35] This means that for every new iteration, or additional hardware supported new code has to be written that is especially designed for the specific hardware. HALs provide specific services to the upper software layers including the integration with the ANSI C library, device drivers, a defined and standardized HAL API, system initialization tasks and device initialization tasks. [35]

The idea is to use the concept of the HAL and apply it on the given problem of MissionEDU. This means to create an abstracting software layer providing defined services to the upper layers in order to enable platform independent access of the robotics system.

4.2.2 ARS Design

Based on the design of the Hardware Abstraction Layer, the MissionEDU ARS has to be created for every platform that requires support. By default it is automatically generated code created from a JSON config file called CDEF. This automatic code generation happens during the configuration process. If a configuration client connects, the current ARS is shut down. The configurator then is able to add or remove functions and libraries to enhance or change functionality. After saving the configuration the ARS is compiled and launched as soon as the configurator disconnects. If the compile fails, the old state is reverted and the configurator notifies the user.

Since the ARS is necessary for the server to provide access to the controller, it is managed directly through the server and provides a consistent interface for the server to access.

It is intended to be replaced by third-party software and enhanced by user configurations. Additionally to the default code generation versions of the ARS in other - or the same - programming languages can be written and compiled to replace the existing.

4.3 Remote Procedure Calling with ARS

The abstracted robotics system gives the clients the ability of remote procedure calls. All available procedures are sent to the clients, which execute them through a message specified in the MiRACLE protocol. Called procedures are received by the server and forwarded to the ARS which then executes the code behind. In order to enable independent communication between the ARS and the server a communication interface called ARS-CI was developed.

4.3.1 User Defined Methods (UDM)

To increase customizability and it's usability users can add their own customized functions and methods, which are provided to the user of the programming client. A UDM can return values of the data types integer, double and bool. Methods can also accept one parameter that can be integer, double or bool. The design of the UDMs is related to the factory pattern in software design since they are generated on demand by a generating structure. They, however, differentiate in the way that with a normal factory patterns, the methods are generated in runtime as requested while they are generated for MissionEDU during the compile process.

4.3.2 ARS Communication Interface (ARS-CI)

The ARS communicates over UNIX-Domain-Sockets (UDS) utilizing a standardized yet minimal protocol. This makes it possible to replace it with an implementation created in another programming language, if in example, the robotics controller's library doesn't support C++.

The ARS launch procedure consists of creating a UDS in a dedicated thread of the server, launching the ARS and the connection process in which the ARS connects to the UDS created by the server. After that the server can communicate via the ARS-CI protocol

with the ARS.

ARS-CI protocol includes 5 message types:

- Connection message
- Disconnect message
- Request message
- Result message
- Error message

The messages of the ARS-CI consist of 2 ASCII formatted values, separated by the newline character. The first describes the message type, the second the attribute.

4.3.3 Controller Definition File (CDEF)

In order to store the information necessary for the automated build process and to access information for the remote calling CDEF files are used. CDEFs are JSON formatted and contain:

- a description of the compiler (including compiler flags)
- all libraries that have to be included
- the code of the user defined methods,
- its return types and parameter types
- its parameter ranges and the code generator program.

Separating the generation of the ARS into a completely self-contained program helps with increasing support for different platforms, since the code generator can be maintained isolated from the rest of the server.

4.3.4 ARS-GEN and Third Party Implementations

By default a build and configuration tool called ARS-GEN for C++ programs is supplied. It translates the CDEF file into a C++ program using a template file in which the communication interface is implemented. The UDMs are generalized using wrapper methods. Pointer to these functions are stored in a function-pointer-map data-structure. The methods' name that is included in the request is used to access the corresponding method.

Chapter 5

Server

Author: Daniel Swoboda

Servers are computer programs designed to provide services of various kinds to clients over standard means of network communication. In MissionEDU the purpose of the server is to provide the clients with data, to enable communication with the abstract robotics system and to configure the setup.

5.1 Overview of Server-Client Systems

5.1.1 History of Servers



Figure 5.1: An IBM 7090 at the NASA mission control computer room taken during Mercury Atlas 6 in 1962 [36]

The modern server-client approach originates from the mainframes (example depicted in Figure 5.1) of the 1950s-1970s, which were computers providing services to one or more connected terminals. Programs were executed on the mainframe while the terminals served

as an interface to the operators. With the dawn of PCs and Local Area Networks in the 1980s mainframes evolved into servers that provide specific functionality to clients instead of performing all the tasks. The clients on the other hand replaced the terminals and are now able to do their own operations without the server. [37]

5.1.2 Modern Servers

The server-client concept evolved from pure networking applications to also be used in inter-process communication and local system design. Not only modern operating systems with micro-kernels like macOS rely on servers to provide functionality to clients (e.g. GUI window management), but also modern applications also use one or multiple background-processes that provide functionality to a client enabling the interaction with the system [38]. Modern servers are designed to be lightweight, efficient and scalable.

5.1.3 Server-Client Systems in Robotics

In robotics applications there are two common use cases for server-client designs. First, the remote programming environments and second the software-frameworks for robot control and component access like ROS.

Remote programming environments allow a developer to write code that is directly saved and compiled on the robotics controller. The editor can be a standalone desktop application, a phone app or a website (either hosted on the robot or on another device). Such programming environments are commonly found in educational and hobbyist robotics. [39]

Robot control and component access software-frameworks on the other hand are systems that run locally on the robot. They allow the usage of a modular design through hardware abstraction and a common protocol. Every hardware component is connected to the main server (which is responsible for program execution) via the network through a software client. The client includes hardware-specific code to access the component. It also has to speak the protocol for integration with the server. Such a system is the core of the Robotics Operation System (ROS). [40]

The MissionEDU server fits into both categories. While it provides access to the hardware through a standardized interface, it also enables the communication to the remote programming and configuration environments (MissionEDU Client and MissionEDU Configurator).

5.2 Conception

In order to improve the design of the server and to choose the right tools and programming languages, some of the most important aspects were analyzed and conceptualized in detail. The server acts as a proxy in the sense of the proxy pattern to the connected clients and the ARS [41].

5.2.1 General Design

The MissionEDU server was designed with the software setup of common robotics controllers in mind. These often feature a Linux operating system and specifications similar to more recent smartphones (compare [18] [19]). A server running on a robotics controller should therefore limit its resource usage to a minimum in order to not interfere with the robotics system, especially when these systems might perform heavy tasks like computer

vision. It should also be compatible to most controllers by default or with a minimum of additions to the system.

5.2.2 Passive Design

To reduce the load on the system the server should be designed to be in a passive state while idling. A server with such a design only awakens when an event occurs. In MissionEDU such a events are triggered by messages that are sent either by the connected clients or the ARS. Because of the MiRACLE protocol's state transition model the server only changes state from idle to active state on request from a connected client. When a message from the client is received, it is either forwarded to the ARS (the client then awaits the completion of the execution in idle and sends the result to the client) or performed by the server itself (which also sends the result to the client), depending on the message type. By doing so, most of the strain is put onto the robotics system itself which is designed to perform tasks for longer periods of time.

5.2.3 Usage of ARS-CI and Choice of Language

The server also had to work with ARS-CI which means it has to support UNIX-domain-sockets. Because of that a programming language with direct and integrated access to UNIX APIs were preferred. This eliminates languages like C# which can be compiled under Linux but lack UNIX-API interfaces. The language also was required to support a kind of socket API (preferably the Berkeley socket API) so that swockets could be easily implemented in that same language.

5.2.4 Library Support and Ease-of-use of the Programming Language

The programming language was required to feature extensible support for libraries and an active community of developers adding additional features to the language. Additionally it also had to be be easy to use and write in order to increase maintainability and platform support.

5.2.5 Installation

MissionEDU Server has to be easy to install and support auto-start as a daemon to increase user-friendliness. A installation tool has to be easily producible for every platform that is intended to be supported.

5.3 Implementation

The server was implemented according to the conception. The chosen language is Python 2.7 which is an interpreted language commonly used in a variety of open-source projects. A variety of APIs and libraries were included in order to decrease implementation time.

5.3.1 Programming Language

Python is an interpreted multi-paradigm language instituted in a variety of scenarios from networking applications, to web servers, robotics and even aeronautic systems. It features dynamic typing and automatic memory management. Since Python is open-source, its reference implementation CPython is free and open software developed using a community-based development model. The language was conceived in the 1980s by Guido van Rossum,

who started working on the implementation in December of 1989. Python 2 was first released in 2000 with 2.7.2 being the latest version (released in 2010). Python 2.7 is the last major release of the Python 2.x series and is considered the most stable and widely supported Python version. [42][43]

5.3.2 APIs and Libraries

- To handle network communications and manage connected clients the `swockets` library was used. It consists of a single file that contains all the modules that have to be included.
- For communication with the ARS the CPython `stdlib` module `'sockets'` was included to establish a UNIX-domain-sockets communication. It is a wrapper of the C call to the UNIX API.
- To serialize and deserialize JSON the CPython `stdlib` module `json` was selected. CPython JSON can be replaced with faster third party implementations. This, however, increases the installation complexity.
- To launch and manage the ARS as a subprocess the CPython `stdlib` module `subprocess` was used. It allows to call any executable binary on the system as a managed subprocess.

5.3.3 Event Handling

The server features threaded and loop-based event handling. One main thread handles the communication with the ARS as well as the parsing of command-line input while multiple threads manage the network connections. This design is a result of the usage of `swockets` as the underlying architecture. Instead of having a separate class as the event handler, the same class was used for all main features of the server. This means that the behavior of the server is controlled majorly by the `swockets-events`, which is intended behavior.

5.4 Performance Analysis

The performance of a program is an important indicator for the effectiveness of the code. Ineffective programming often leads to higher resource demands. Analyzing the performance is especially important in low-end devices. While modern robotics controllers are getting more and more capable, they are still considered low-end devices when to desktop PCs or notebooks. [44]

5.4.1 Reasons for Analysis

This analysis was made in order to gather information about runtime behavior of the implemented software and to search for ways it might affect the robotics system, the hardware itself and the battery lifetime. These indicators are critical to the adaption of MissionEDU.

5.4.2 Methods of Analysis

There are 3 kinds of analysis tools to monitor program performance: Statical Analysis, Dynamic Analysis and Hybrid Analysis.

Static analysis tools are mechanisms for performance evaluation that do not modify the binary of an application. Therefore they rely on techniques like source code instrumentation and sampling of indicator values from outside the software. Static analysis tools only work with data that is externally available. Recorded results are analyzed using statistical methods to determine performance behavior. Since these tools are additional programs running on the device they add overhead and can cause system slowdown. [45]

Instead of the black-box view of the binary of a program statical tools are confronted with dynamic analysis tools rely on additions and alterations to the program to collect data. These tools either rely on manually made modifications to the program or are capable to inject customized analysis routines into the binary of an application. By making use of the information provided a high level of granularity is provided. A tradeoff of this method is that the analyzed program itself is modified. Because of compile and cpu optimizations this modification can have a big effect on performance. [45]

Hybrid analysis tools use a combination of both techniques to gather comprehensive performance data on multiple levels. With this approach runtime is increased to a level that some tools cannot be combined. [45]

To analyze the performance of MissionEDU Server a simple sampling tool was written to log the memory and CPU usage of the program every second. This method was chosen because of its simplicity and cross-platform availability. The logs were than analyzed using statistical methods implemented with numpy and scipy.

5.4.3 Tested Devices

For detailed performance analysis on the target devices of MissionEDU Server the robotics controller KIPR Wallaby was chosen. The Wallaby features a single-core ARMv7 processor rev2 with a single-core CPU, a clock-rate of 750MHz and 512MB of RAM. For comparison a Notebook by Apple with an Intel Core i7 quad-core CPU, a clock-rate of about 2.3GHz and 8GB of RAM was used as a reference device.

5.4.4 Tested Indicators

Two indicators were chosen to be monitored and analyzed: memory usage and CPU usage. These two values can be accessed with the tool 'ps' on Unix and Linux systems [46]. A simple bash-script (listed in Code 5.1) was used to track the usage of these resources every 0.5s.

Code 5.1: The script used to track the indicators

```

1 #!/bin/bash -e
2 echo "performance tracking script"
3 NAME="$(date '+%Y-%m-%d--%H-%M-%S').csv"
4 touch $NAME
5 echo "date;time">$NAME
6 while true; do
7   DATE=$(date '+%Y-%m-%d;%H:%M:%S')
8   CPU=`ps -p $1 -o %cpu | tail -n +2`
9   MEM=`ps -p $1 -o %mem | tail -n +2`
10   echo "$DATE;$CPU;$MEM">>$NAME
11   sleep 0.5
12 done

```

The indicators were tested in idle mode, under minimum load (normal execution of commands) and high load (compilation of the ARS). The impact on the systems then was analyzed.

5.4.5 Idle Performance

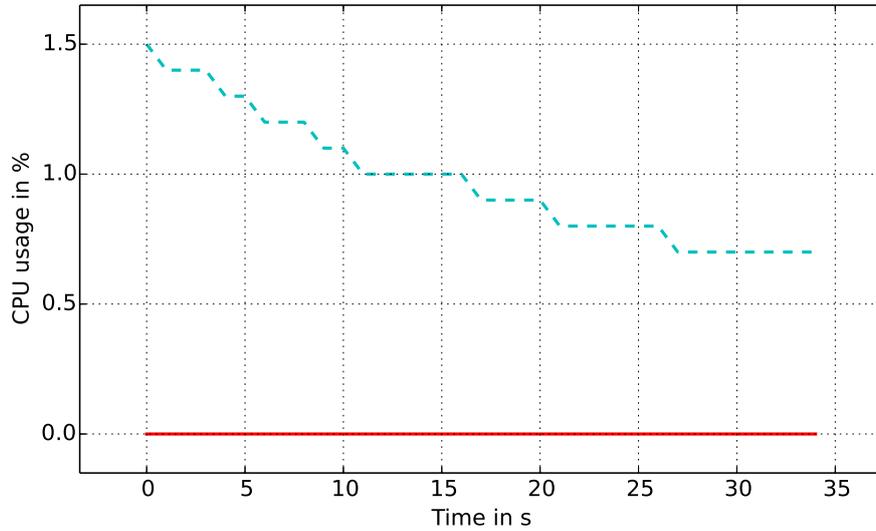


Figure 5.2: Performance in idle, red-straight line is the reference notebook, blue-dashed line is the Wallaby

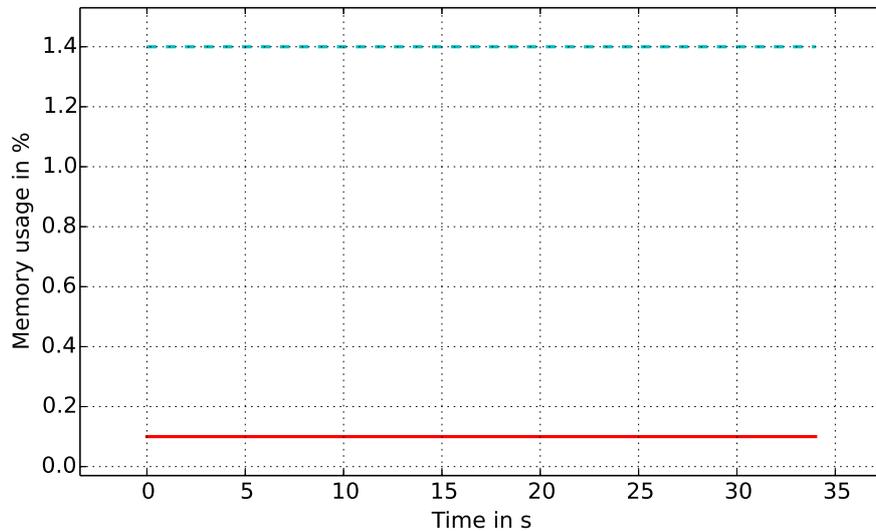


Figure 5.3: Memory usage in idle, red-straight line is the reference notebook, blue-dashed line is the Wallaby

Figure 5.2 and 5.3 depict the resource usage by time as line diagrams in an idle situation. There were two servers set up. One of the servers was running on the reference notebook while the other one was running on a Wallaby controller. In this test no clients were connected.

The results of the memory analysis (Figure 5.3) 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 mode. The overall usage in percent was bigger on the

Wallaby because it has less memory.

CPU performance (Figure 5.2) stayed constant on the reference machine. 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 of this behavior might be optimization of the execution either by CPython, the operating system or the CPU.

5.4.6 Regular Load Performance

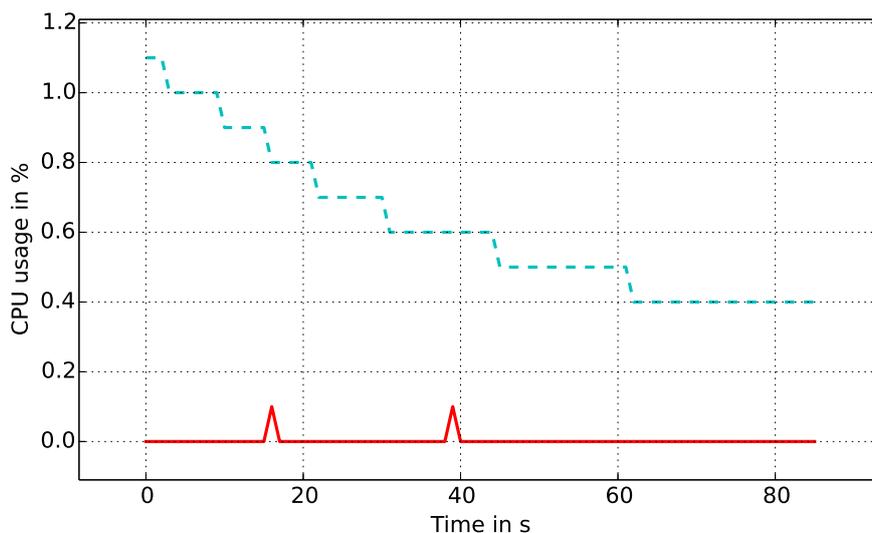


Figure 5.4: CPU usage with a client connected that regularly performs a request, red-straight line is the reference notebook, blue-dashed line is the Wallaby

The purpose of the second experiment was to analyze performance under regular load with one client connected. To synchronize the recorded data two clients were used that sent requests at the same time. The clients sent two requests for simple mathematical operations implemented in the ARS each.

While memory usage was once again constant and at the same levels of experiment 1 (depicted in Figure 5.3) the CPU usage differed. In this scenario the CPU usage of the server on the wallaby started at 1.1% which is 0.4% less than in the first experiment (depicted in Figure 5.4). The decrease of CPU usage over time was observed again while the reference notebook data shows two peaks at the time of the requests.

5.4.7 Compile Performance

Performance of the server while compiling was similar to the performance in the second experiment. However, the compile process added additional strain to the devices.

While the compilation took 10s on the Wallaby, it took a mere second on the reference notebook. During this time it cannot be recommended to do other tasks on the Wallaby since CPU usage peaked at 97.4%.

This - however - cannot be counted as CPU usage of the server since it was created by the C++ compiler instead.

5.4.8 Discussion of Results

The results of the analysis show that the server's usage of memory and CPU can be considered lightweight with a memory footprint of 9.8MB and less than 2% of constant CPU usage on the Wallaby and even less on the reference device. With this results any effects on other programs running on a controller can be precluded. It can be assumed that the server is able to run on even less powerful systems without a problem.

5.4.9 Methods to Increase the Performance

Since Python is an interpreted language an underlying piece of software that translates the code in real time called interpreter is needed. It reduces the performance of the programs since it adds additional layers of code that have to be executed. To further increase performance on less powerful systems a Python compiler can be used. Such a compiler would translate the Python code into binary machine code which is directly executed without the need of an interpreter.

5.5 Compatibility

Since cross-controller compatibility is one of the main goals of MissionEDU Server high levels of compatibility were one of the most important goals while developing the server.

5.5.1 Minimum Requirements

The minimum requirements found to be were:

- Linux, macOS and most other Unix OS
- 350 MHz CPU
- 20 MB of free Memory
- Python 2.7 support
- C++11 compiler (for ARS compilation)

5.5.2 Compatible Systems

Compatible systems which fulfill the requirements and were tested during the development of this thesis:

- KIPR Wallaby
- KIPR Link
- Raspberry Pi
- PRIA Hedgehog
- Intel Edison

5.5.3 Compatibility Issues

Some Linux distributions don't come preloaded with a Python 2.7 install. For these distributions it has to be either compiled on the device or a precompiled version has to be installed. Other distributions don't include the full Python stdlib. The stdlib has to be supplemented in order to run MissionEDU.

Chapter 6

Configuration Client

Author: Daniel Swoboda

Since editing files in a custom format on a remote device without a graphical user interface is not considerable for the intended usages of this project a user friendly approach with a graphical configuration client was developed. The configuration client is written in C++ using the Qt framework and the swockets library for networking. It's expected environment is a standard Linux or macOS PC.

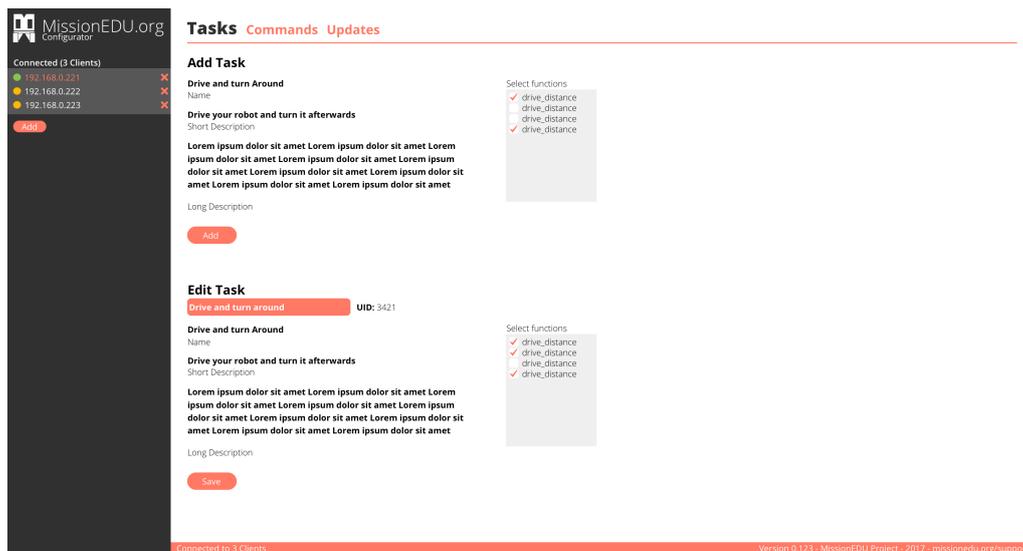


Figure 6.1: MissionEDU Configurator third concept

6.1 Qt framework

The Qt (pronounced cute) framework is the most widely used open-source cross-platform framework for graphical user interfaces (GUI). It supports a variety of different language through bindings but is originally developed for and in C++. Since the application should run on any PC without the requirement of additional packages or programs like in the case of scripting languages, C++ was chosen as the language to go. Since MissionEDU is open-source Qt was available for free use. [47]

Qt was first developed by Haavard Nord and Eirik Chambe-Eng and their company

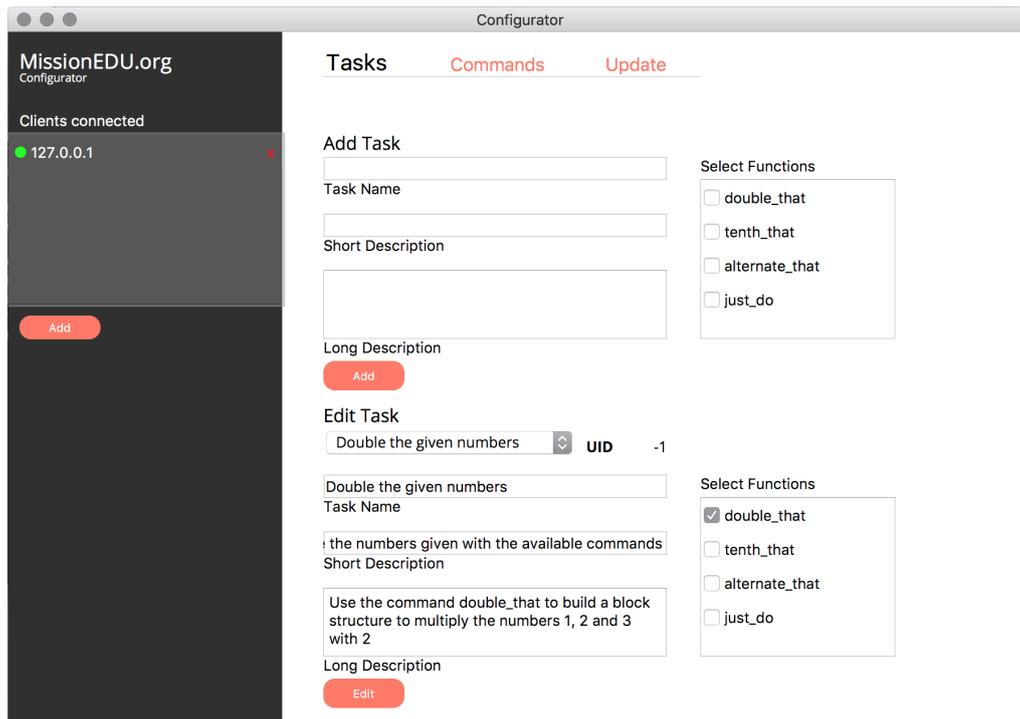


Figure 6.2: MissionEDU Configurator in task configuration view

Trolltech. It was later acquired and open-sourced by telecommunications company Nokia and now is developed by the Qt company. [48]

Qt extends C++ with functionality for signals and slots. Signals and slots are the Qt approach to event handling for GUI elements. A signal is the event created by interaction with the element, while a slot is the function that is called when the event is triggered. Qt also adds features for XML support, database access and localization, which were not needed. [47]

6.2 Design and Functionality

MissionEDU Configurator features a simple GUI without any context menus or hidden elements. This helps flattening the learning curve since all elements are already visible within a view. To reduce the complexity, the amount of elements visible at once was reduced with the use of 3 views. The elements were grouped by their logical meanings. Additionally for elements that should be directly available all the time a sidebar was added.

As shown in Figure 6.1 the general layout wasn't changed since the third concept. The sidebar includes a list of all the connected clients, identified by their IP address. It also features a button used to add a new connection and buttons to close the connection to each of the clients.

To add, change and remove tasks graphical elements are collected in the task view as depicted in Figure 6.2. In the add task section a new task can be added to the controller. A name, a short description and a long description can be added and the functions needed to solve the task can be selected. In the edit task section existing tasks can be adapted.

To add new commands to the ARS and edit existing ones the command view depicted in Figure 6.3 can be used. It features drop down menus to select return and parameter

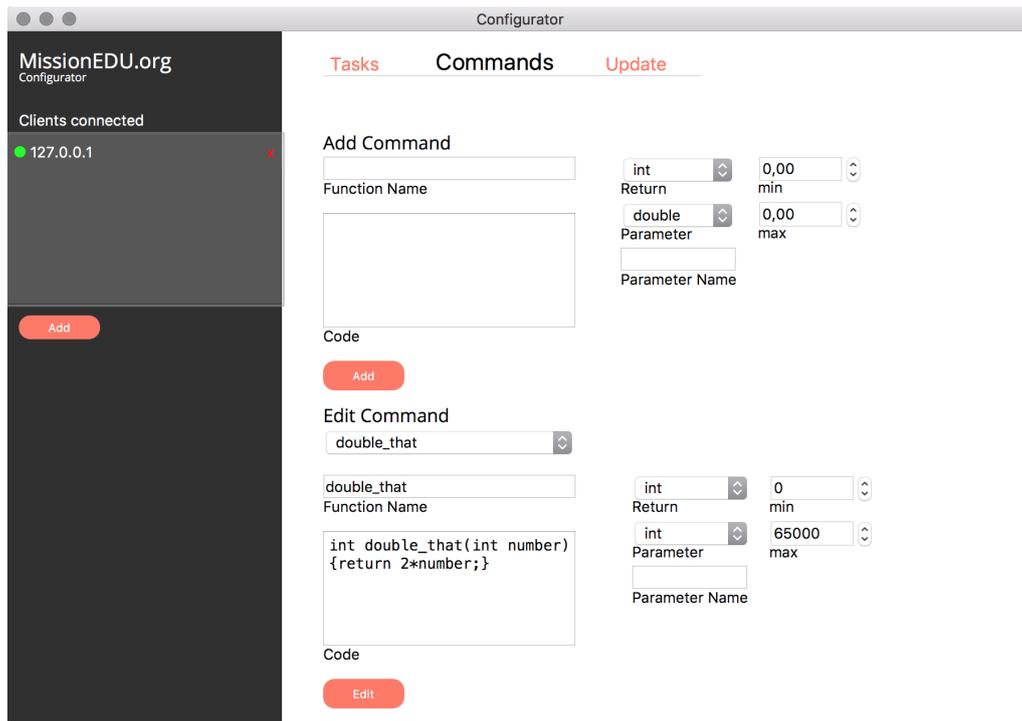


Figure 6.3: MissionEDU Configurator in command configuration view

data types, a field to set the function name and a code editor to edit the code behind the function.

6.3 Implementation

The Configurator was implemented in C++14 with focus on usage on Linux and macOS. Windows specific changes to increase the compatibility were not made. It was decided to use the platform native look and feel for the UI elements but to use a custom theme for the buttons and the sidebar as well as a custom font. By doing that the design of the MissionEDU website, client and configurator all have a related look accompanied by the same color scheme.

Chapter 7

Robot Programming Interface

Author: Markus Pinter

The robot programming interface is the interface through which the robot is programmed, either by directly programming the hardware or by using higher-level programming languages.

7.1 Overview of Robot Programming Interfaces

Since robots left the industrial environment and entered a broader field of applications, the demand for technicians in this area increased. To tackle the absence of qualified engineers, companies tried to develop programming interfaces that are simpler to use. By doing that more people are able to learn the usage of such a system in a shorter amount of time. According to Geoffrey Biggs and Bruce MacDonald, a robotics programming system may use automatic programming or manual programming. For both of which software architecture plays an important role [39].

The software architecture describes the underlying part of the program, over which access of the hardware parts is provided.

Automatic programming doesn't rely on programmers writing code, it rather uses a system by which a robot is physically taught which movements it has to perform. Because of that automatic programming tools are not used in educational robotics, since it focuses on teaching programming skills to students.

Manual programming can be divided up into two different methods, Graphical programming, and textual programming. Recently textual programming is used, because this method provides high flexibility and control [39].

7.1.1 Botball Programming Interface

The Botball environment only supports textual programming languages C and Python. The programming interfaces run on the KIPR Wallaby controller as a web-service. Projects containing the code are created through the web-app whilst the data is saved in the controller's memory. The Code is edited through the interface which is shown in Figure 7.1. In order to access sensor data and control actuators, the "libwallaby" library is used. A console provides an endpoint for the output stream of the executing program.

Users don't need to have a local copy of the code because the programs are stored on the controller itself.

The design is pretty modern and comfortable to work with. At the top, all important operations for programming are displayed. At the right, all projects created on the Wallaby

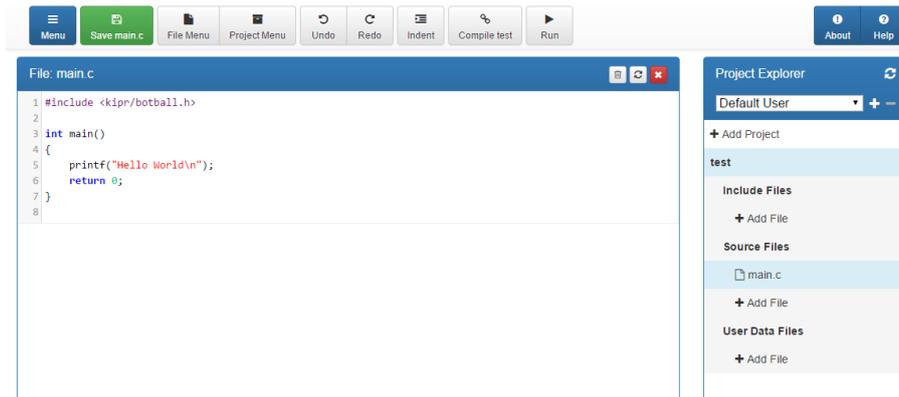


Figure 7.1: KIPR Software Suite

are shown and can be selected at any time. This layout makes it simple to use and the user doesn't have the feeling there's too little space for code display.

7.1.2 LEGO Mindstorms Programming Interface

One set of controllers that is broadly deployed in educational robotics is the LEGO Mindstorms Series. It is usually programmed in the NXC language. NXC can be used with the Brick Command Center (BricxCC) which is shown in Figure 7.2. This interface still uses Windows XP Application style and looks rather complicated. There are too many buttons at one place, which could overwhelm students.

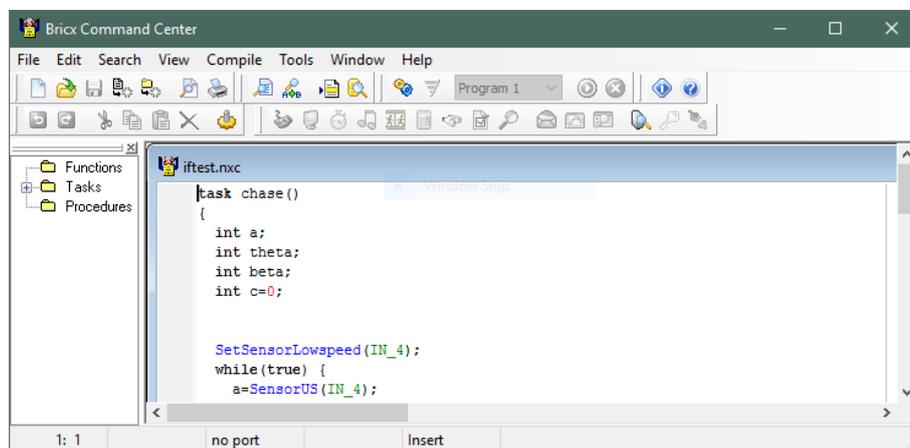


Figure 7.2: NXC Program displayed with Brick Command Center

The LEGO company provides a graphical programming language and a corresponding programming interface for all of their existing kits. In Figure 7.2 five buttons are depicted in the navbar at the top which makes it simple and clear. When dragging a new Element from the bottom to the current workflow in the middle of the screen, a clipping mechanism clearly shows where it is going to be inserted when being dropped. The big flaw is the complex design of the individual elements because too many parameters are displayed at a single time and one doesn't really know what their functionality is good for.

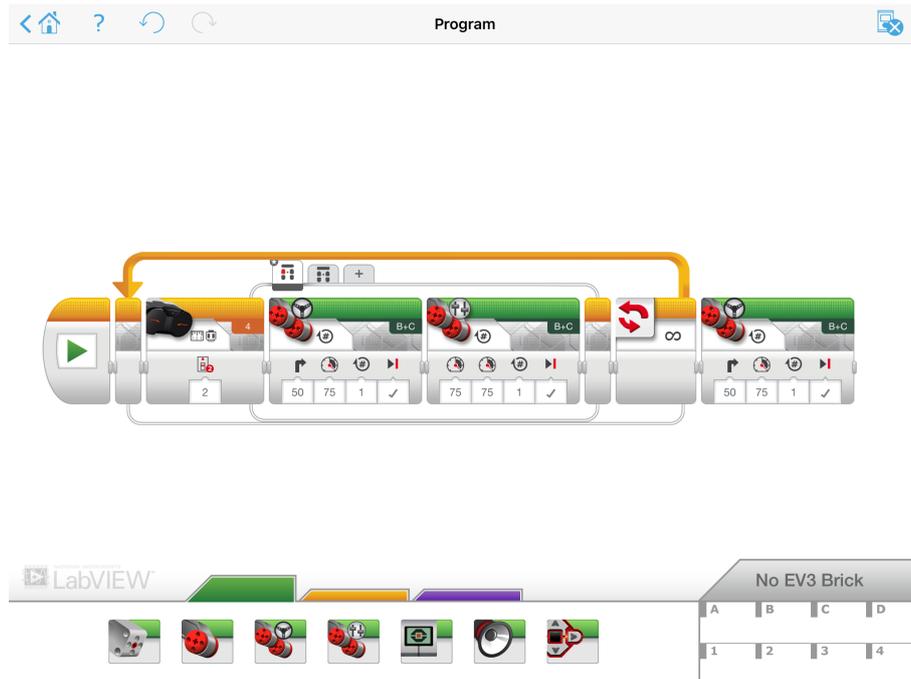


Figure 7.3: LEGO Mindstorms iOS app

7.1.3 Arduino Programming Interface

Arduino is a very popular robot controller not only used by hobbyists but also applied by large companies. Many different programming interfaces exist for various use cases. The most known is the Arduino IDE. The design is very modern and lightweight. Most of the functionality is packed into the menu bar, thus only the most important operations are displayed as buttons. An additional window can be opened to display the console when the program is executed.

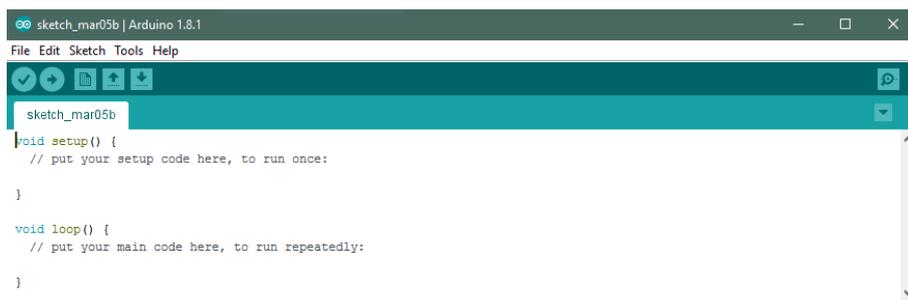


Figure 7.4: Arduino IDE

7.2 Design Strategies in Mobile Applications

Since Apple released the AppStore for iPhone's in 2008, mobile application development increased drastically. Mainly two different design patterns are used across smartphones and tablets.

7.2.1 Skeuomorphic Design

A skeuomorph may be described as an object or feature which imitates the design of a similar artifact in another material or technique. It may also be defined as an element of design or structure that serves little or no actual purpose of the product in the new material but was essential to the object being made in the original material [49].

In mobile applications, this design pattern was often used by apple for apps until iOS7. Since the purpose of this design is to imitate an existing physical object, like a calendar, it gets clear and easy for the user, how the individual application needs to be used.

7.2.2 Flat Design

Since Microsoft released Windows 8 this design pattern got much attention. While being criticized by many users on pcs and laptops, flat design got increasingly more popular on mobile devices. It's characterized by its simplicity. Rectangular shapes with no gradients or rounded edges are typical.

7.3 Graphical Element Design for VPLs

Since interaction between the user and the device directly takes place over Graphical Elements, they are the most important part of a Visual Programming Language. The better the design the easier each program can be understood. Design variations can be broken down to the main pattern.

7.3.1 Vertical Linking

This approach builds up program flow vertically on the screen and therefore is read like a normal textual program from top to bottom. Usually, elements that only get executed when a condition is fulfilled like in the case of an if or while statement gets indented to increase readability.

When switching from a visual programming language to an textual programming language afterward, this design is a huge advantage, because students already got that feeling of reading statements top-down.

7.3.2 Horizontal Linking

Program flow is constructed row-wise and read from left to right and therefore is a more abstract version of programming. Since blocks of statements cannot be intended in this type of design, they are either colored differently or shrunken down. Clearly, coloring is the better approach when supporting nested blocks because otherwise they would be too small to be read at some point.

Most students were only conflicted with this type of reading and writing, thus it is easier early on to get familiar with programming.

7.4 MissionEDU Programming Interface

The programming interface of MissionEDU is the most important part of the client since the programming of the robots is thus executed. Hence, it needs to be as simple to use and as easy to learn as possible. Designing the app was approached in a way, that text isn't

used to discriminate between elements of the GUI, rather different colouring and shape signals users their functionality.

Since most modern devices use flat design nowadays, this pattern was also chosen for this project.

The programming interface splits the screen vertically into two spaces. On the left side of the screen, the so called workbench, which represents the current program, is displayed. On the right side the element container, which contains all programming elements, can be seen. They are divided into logical sections to provide a better overview.

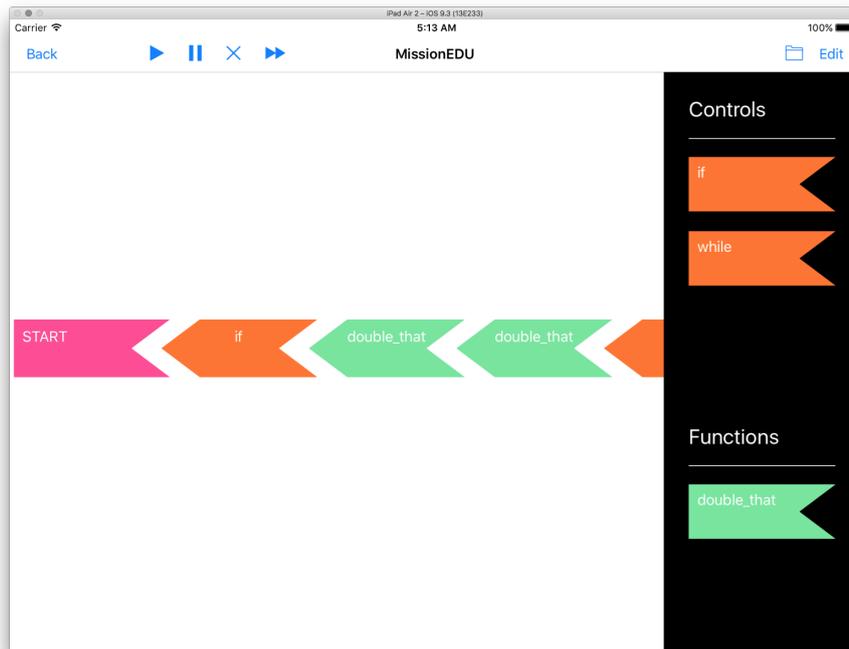


Figure 7.5: MissionEDU Graphical User Interface

7.4.1 Programming Elements

MissionEDU is majorly developed for students with no programming experience, so a Horizontal Construction approach was chosen. Four different types of elements exist.



Figure 7.6: Variations of Elements

- The Begin Block is used at the beginning of each row.
- The End Block stands at the end of a row.
- Mid Blocks are displayed between the above.

- Parameter Blocks are used when parsing return values of other programming elements as arguments for the current element.

7.4.2 Navbar

The navbar is positioned at the top of the screen, as shown in Figure 7.5. If users wish to switch these two spaces, they can do so in the settings displayed on the navbar.

In the navbar students can:

- execute the program
- pause or stop the program
- step forward to the next element
- switch between programs
- navigate to settings or title screen

7.4.3 Programming Element Container

It contains all available programming elements that can be used. They are divided into sections for a better overview. If there are too many elements the presentation of all programming elements simultaneously is not possible. Hence, students can scroll through the container.

7.4.4 Workbench

This section displays the program flow of the program students are currently working at. To add a programming element to the sequence, the desired one can be dragged from the container into the workbench at the demanded position. A program always starts with a start block where every other statement needs to be appended. When inserting an if statement or while loop a second element automatically gets created to enclose their inner elements. The workbench can be scrolled dynamically on both axes when space on the display isn't sufficient.

7.5 Tasks

Tasks are the essential part of the MissionEDU concept. It is an objective the user has to achieve in order to move forward. It is created by the administrating user, who can determine which programming elements are allowed to use and which are not. Internally tasks are represented as JSON objects with a format shown in Code 7.1.

Code 7.1: General Task structure

```
1 {
2   "taskUID" : "<uid>",
3   "taskName" : "<taskName>",
4   "taskDesc" :
5   {
6     "shortDesc" : "<shortDesc>",
7     "longDesc" : "<longDesc>"
8   },
9   "commandsAvailable" : [<list of allowed commands>]
10 }
```

Each task gets automatically an unique identifier (UID) on creation. In addition, a task name and a task description, to explain which goal the students need to achieve, is provided. The list of commands which are allowed to be used is included.

7.6 Programs

"A computer program is a collection of instructions that performs a specific task when executed by a computer [50]." In MissionEDU a program is a sequence of commands. A command is an equivalent to a programming element, with the exception that the term programming element is only used in relation to the graphical user's interface. Multiple programs can be written to solve one task. They are saved on the client and therefore won't sync to other end devices. One workbench represents one program.

When choosing a task, a program automatically is created. If a new empty workbench is desired, the user taps on the program menu button on the navbar, then on the plus button in the pop-up menu.

Programs are saved in the MateScript-format. When choosing a task, the client loads available MateScript-files into the GUI. When the app is closed, all files are saved on the client device.

Chapter 8

MateScript

Author: Markus Pinter

MateScript is a lightweight textual scripting language that can be used to write programs and save them.

8.1 Programming Language Design

According to C. A. R. Hoare the most important aspects of a good programming language are simplicity, security, fast translation, efficient object coding, and readability [51]. The main focus of MateScript lies on simplicity, fast translation, and readability. This programming language was designed for two reasons. Firstly as a tool to convert a graphically constructed program into a textual version, which then can be executed on a device. And secondly as a way to guide students one step closer to programming languages used in the industrial and research industries.

MateScript syntax resembles mostly to C with few exceptions. Since this programming language can evolve over time, only version 1.0 with its features are discussed.

Code 8.1: Simple MateScript program

```
1 Drive(16); //executes function called Drive with argument of 16
```

As opposed to most programming languages, a MateScript-Program starts at the first line of the file and is executed sequentially. Hence, it has no specific entry point.

MateScript also supports the functional conditions `if` (with an optional `else` block), `while` and `count`. After one of these keywords a condition is expected which determines whether the surrounded block is executed or not.

8.1.1 Declarations, Definitions and Function Calls

When declaring a variable, it doesn't have an assigned data type. As soon as using a variable, its type is set to the type of the value. Declarations, definitions and function calls conclude with a semicolon and parameters get parsed in parentheses after the function name, shown in Code 8.1. Parameters can be constant values or another command with an appropriate return value.

Defining a variable means either assigning a value or also declaring it by using an equal sign.

Functions directly correlate to commands getting executed on MissionEDU-Server. Thus functions cannot be defined due to the architecture of this project and therefore would be an unnecessary feature.

8.1.2 Types

Types play an important role in the evaluation process of expressions, as well as in parsing arguments to a function. Results of functions also have a defined type.

Following are supported Types of MateScript: int, float, double, boolean, void.

Code 8.2: Variable Declaration

```
1 integerValue = 10;
2 floatValue = 10.0f;
3 doubleValue = 10.0;
4 booleanValue = true;
```

As illustrated in Code 8.2 variables automatically get an assigned type unlike in C. Float values must contain exactly one dot and an extending f at the end. Doubles are declared just the same way, but without an extending letter. Boolean values can only be assigned by true or false.

8.1.3 Logical Expressions

"In general programming expressions can be defined as variable names, function names, array names, constants, function calls, array references, and structure and union references. Every expression combined in any way with another also results in an expression [52]."

Logical expressions in MateScript are defined as a combination of expressions evaluated as a boolean.

Code 8.3: If condition

```
1 if TophatLeft() <= 300 && TophatRight() > 300 {
2     Turn(16);
3 }
```

No parentheses are used to surround the expression. Conditions need to resolve as a boolean value to be accepted. This can be achieved by using operators, which require one or more expressions as described below.

Supported operators are:

- && returns true if both expressions return true
- || returns false if both expressions return false, else it will be true
- < returns true if the expression on the left side has a smaller value than the one on the right
- > returns true if the expression on the left side has a greater value than the one on the right
- <= returns true if the expression on the left side has a smaller value than the one on the right or both are equal
- >= returns true if the expression on the left side has a greater value than the one on the right or both are equal
- ! negates the expression, needs to be typed before the expression

8.1.4 If Statement

If statements belong to the family of control statements and are used to execute certain code snippets only when a specific condition is fulfilled. The condition is an expression and therefore has to be evaluable either to true or false. Statement blocks must be written in curly brackets as shown in Code 8.4.



Figure 8.1: Graphical if statement

A graphical representation of the if statement as used in MissionEDU is depicted in Figure 8.1.

Code 8.4: Structure of if statements

```
1 if expression { //code only gets executed when expression is true
2 } else {
3 }
```

In Code 8.4 the structure of an if statement is illustrated. After the expression curly brackets are mandatory to compile successfully. Every if statement may be appended by an else block also surrounded by curly brackets.

8.1.5 While Loop

While loops are also control statements and used to execute a code snippet as often as a certain expression is true. Curly brackets are required to surround its content.

Code 8.5: Structure of while statements

```
1 while expression { //code only gets executed as long as expression is true
2 }
```

8.1.6 Count Loop

A special type of loop implemented in MateScript is called count statement. Its purpose is to execute a code snippet a specific amount of times.

Code 8.6: Structure of while statements

```
1 count int { //code gets executed as often as the number provided
2 }
```

Count requires an integer value to know how often its content needs to be executed.

Chapter 9

MissionEDU Client

Author: Markus Pinter

The proposed MissionEDU Client is used to enable people to work with MissionEDU in an efficient manner. It consists of a Graphical User Interface (GUI), a Scripting Engine and a Network Interface.

9.1 Overview of Client Frameworks

Nowadays many client frameworks for Application Design are available. They were created for different purposes on a variety of platforms. The most popular frameworks are programmed object-oriented.

9.1.1 WPF

WPF or Windows Presentation Foundation is a framework developed by Microsoft exclusively for windows platform. The specialty of this framework is an Extensible Application Markup Language (XAML) where graphical elements can be easily created and modified. In addition, elements can also be created in C#. Bindings enable the interaction between XAML and the code behind [53].

9.1.2 Mono

Mono is an open source approach for cross-platform development in C#, led by Xamarin. Its real strength is the ability to use Microsoft's .NET Framework on other operating systems than Windows [54].

9.1.3 Cocoa

Cocoa is an application framework made by Apple. It only supports macOS. For iOS, WatchOS and tvOS Cocoa Touch is used, which includes all features required for developing on mobile devices. These frameworks are written in Objective-C and can be accessed via many other programming languages like Swift, Python, Perl or Ruby [55].

9.1.4 QT

QT is a widely used Framework for cross-platform development created and maintained by the QT Company. It's written in C++ and free to use, as long as it's going to be published

under the GNU General Public License (GPL) or GNU Lesser General Public License (LGPL). Supported platforms are UWP, Windows Desktop, Windows Phone, Android, iOS, Linux and macOS

Qt Applications can be developed under Windows, Linux, and macOS. For iOS Apps macOS with XCODE is needed to be able to deploy written software onto the end device.

Independently of the individual platform, applications run natively. Thus, it runs faster than as a web solution. However, QT provides no garbage collection and therefore requires a good understanding of memory management [56].

9.1.5 Cordova

For cross-platform mobile applications Cordova can be considered for usage. It is an open source framework built by the Apache Software Foundation, that provides a possibility to develop native mobile applications with standard web technologies like HTML, CSS, and JavaScript. This gives a wide variety of advantages to working with an already written tool set for Web Apps. In addition, all native services can be used, normally forbidden when working with a browser. Hence the Camera or the Gyrometer of the mobile device can be used.

The Web App runs on an HTML Rendering Engine. Additionally it is able to access camera data, sensor data etc.

The advantage of this technology is the possibility to deploy a Web App natively to a device. Hence, faster execution time can be achieved [57].

9.1.6 Ionic

Ionic is an open source approach for developing mobile applications. It supports iOS, Android and Universal Windows Apps (UWP). Ionic is built on top of Cordova and focuses on designing beautiful applications. Hence, modern and fancy graphics for various user interface elements are provided [58].

9.1.7 Xamarin

Since Xamarin is a subsidiary of Microsoft, it shall not be confused with the projects they created. Xamarin Platform is a framework developed for creating native mobile applications in C#. Xamarin.Android is used for development on Android whereas Xamarin.iOS is used for iOS devices. These methods are designed to fulfill platform specific tasks like rendering. For a more abstract version Xamarin.Forms was created, which enables designing applications for multiple platforms via Microsofts XAML [59].

In addition, a package manager called NuGet can be used to import various plugins made by the community.

9.2 Xamarin for MissionEDU

Since even very platform specific tasks can be implemented or modified in Xamarin, this framework was chosen for MissionEDU project. Additionally, C# is a very reliable programming language the authors of this thesis had the most experience with.

Multiple ways can be taken to use Xamarin in an IDE. Either by integrating the framework into Microsofts Visual Studio on Windows, or by using a standalone application called Xamarin Studio on macOS. Since support for iPad is designated, we considered Xamarin Studio as the more valuable option because XCODE is needed for deployment of

the end device, which only is available for macOS. In addition simulators for both iOS and Android are automatically installed and set up, so the debug process is fast and efficient. As in every other IDE, breakpoints can be set to step through code lines while running in the simulator.

Over the course of the project, it turned out, that compiling the application in Visual Studio wasn't even successful and couldn't be fixed.

9.3 MissionEDU Client Project Structure

Xamarin uses Model-View-Controller architecture, thus MissionEDU was structured in the same way to make readability and refactoring of code easy. Firstly the whole project which is also referred as a solution divided into three sections, the platform independent code and the platform specific code for Android and iOS. Each project has its own controller and view folder to keep things tidy.

Two compiler profiles for each platform were set up.

The Debug profile includes all debugging flags and outputs messages used for error detection. The Release profile disables any outputs and uses fewer binaries, which results in better performance and thus is used for deployment.

9.4 MissionEDU Client Architecture

MissionEDU Client consists of five modules depicted in Figure 9.1.

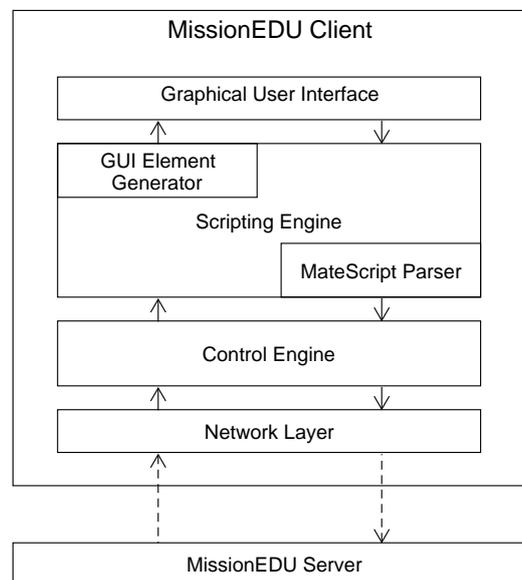


Figure 9.1: MissionEDU Architecture

The graphical user interface is needed to enable an interface between students and the code behind.

The Scripting Engine was created for the following tasks:

- Parsing graphical programming elements to MateScript code
- Generating graphical programming elements from MateScript code

- Parsing MateScript code to an executable format

The GUI Element Generator gets invoked when connecting to a server since a displayable object needs to be created for each command provided by the server. Afterwards a specific element is generated for every program element used in the programming interface.

The MateScript Parser gets called when a program written by the student shall be executed.

The Control Engine represents the core of the client and provides functionality for loading tasks and commands from the server. Additionally, it manages program saving and loading which occurs by switching the program, closing or opening the app.

The Network Layer manages communication between server and client and therefore implements the MiRACLE Protocol.

9.5 Network Layer

The MissionEDU Client has to communicate with a Server in order to be used correctly. The focus of the Network Layer lies in transferring information to the MissionEDU Server through the MiRACLE protocol in an efficient manner. For basic communication Transmission Control Protocol (TCP) is used. Sockets.NET then converts byte stream into JSON objects which are then decoded to be used by the client. For each message type in MiRACLE, a function was created to be invoked by an event handler when the message arrived and was decoded. This event-driven approach is used to execute code for each individual function asynchronously, thus results in a better performance.

Code 9.1: Structure of while statements

```

1 public override bool handshake(Socket sock)
2 {
3     MDebug.WriteMessage("handshake");
4     bool success = true;
5
6     //create MiracleHandler
7     handler = new MiracleHandler(this.ssocket);
8
9     handler.SendRequest();
10    JObject recvObj = ssocket.receive(sock);
11
12    string msgType = recvObj.GetValue("MessageType").ToString();
13    switch (msgType)
14    {
15        case "ACK":
16            handler.OnAcknowledgement();
17            success = GetData(sock);
18            break;
19        case "REF":
20            handler.OnRefuse();
21            success = false;
22            break;
23        default:
24            MDebug.OutputMessage("Message:\n" + recvObj.ToString() + "\n not supported ");
25            success = false;
26            break;
27    }
28    return success;
29 }

```

Code 9.1 illustrates how events get called by the handler during the MiRACLE handshake depending on which message arrived. Decoding is achieved by checking the `MessageType` field in the received JSON object.

9.6 Task Loading

After a successful handshake, the tasks need to be loaded. All tasks get stored into a list, which only exists once during the apps lifetime. This class is called `MTaskList` and provides an interface to store new tasks, get currently saved tasks or search a specific task by its id. The information is sent to the GUI to display each individual task in a readable format.

9.7 Command Loading

Commands play the most important role in MissionEDU. Since commands directly correlate with programming elements of the GUI it is mandatory to be able to generate new ones afterward. A software design pattern by the gang of four was used for this purpose [41].

Commands are converted into `MCommandTemplates` which contain all metadata needed to generate specific commands afterward. These include parameter types, the number of parameters, return type and the name of the command. They are then obtained by the `MCommandFactory`, which generates executable commands.

`MCommandTemplates` are used for generating GUI Programming Elements in the Element Container since they don't get executed.

9.8 Program Functionality

In MissionEDU Programs are collections of commands. On the level of the Graphical User Interface, they are all programming elements in the workbench. In MateScript a program is represented as a code file.

9.8.1 Program Creation and Switching

Since students might have to share tablet devices to reduce overall cost, a support for multiple programs that can be created and switched between is implemented.

Programs can be changed on the programming page by using the folder button on the top right of the navbar. When creating a program, the workbench gets cleared on the GUI and a new empty MateScript file is generated. Hence, switching a program loads the individual MateScript file and generates the programming elements in the workbench.

9.8.2 Program Execution

When executing programs in MissionEDU, the server first has to get informed that a command execution is requested. This is done via the Start Execute Mode of the MiRACLE Protocol. For each command in a program a Send Execute Message can then be sent.

9.9 Graphical User Interface

The graphical user interface was developed in Xamarin.Forms mainly in the XAML format. For generating UI elements at runtime, C# was used.

9.9.1 Start Page

When first opening MissionEDU the start page appears. All new features and updates are listed in the left section. A browser is automatically opened and shows the full patch notes when they are selected.

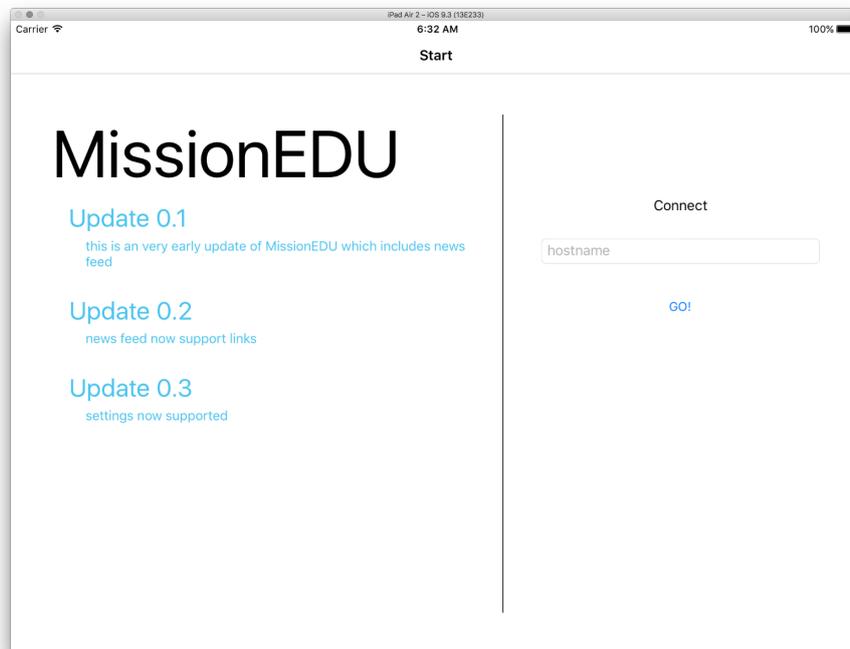


Figure 9.2: MissionEDU Start Page

Additionally, a text box is displayed to enter a hostname or ip address to connect to a valid MissionEDU server. When a connection was successful the application delegates to the task selection page.

9.9.2 Task Selection

This page is based on the master-detail layout, which lists all tasks available and received by the MissionEDU server on the left side and displays detailed information on the right side depicted in Figure 9.3. If the selected task is the requested, the student can tap on the "Start!" button to continue to the programming page and try to achieve this task.

9.9.3 Programming Page

The programming page displays the programming interface and thus the visual programming language itself. All programming elements get loaded before the page appears. Through drag and drop, programming elements from the element container can be easily inserted into the workbench. Parameters can be provided via a dropdown menu or by inserting another programming element. The destined position of the programming element gets highlighted to show students where the clipping mechanism would insert the element.

Figure 9.4 illustrates the layout of this page. It is based on the master-detail approach but doesn't make use of the built-in version in Xamarin.Forms, since the custom design is easier to be managed for draggable UI elements.

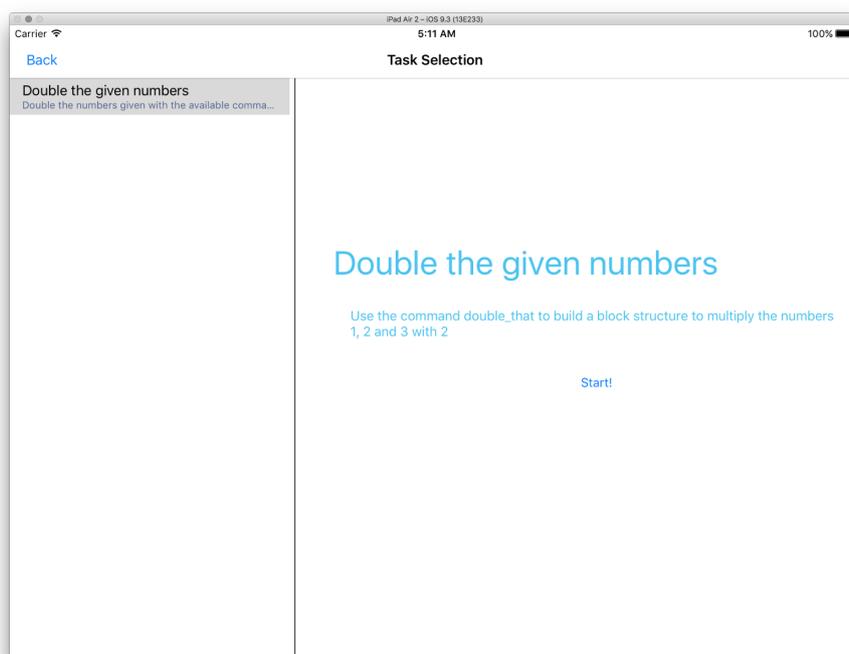


Figure 9.3: MissionEDU Task Selection

9.10 Settings

The settings section is an additional option for students to configure their MissionEDU client for their preferences. Hence, the positions of element container and the workbench can be exchanged so left handed persons can also work as easy and efficient. Other settings like help display can also be managed.

9.11 Help Functionality

The help functionality is a feature of MissionEDU, which shows students an introduction on how the application needs to be used. This is achieved through short pop up messages, which enables an interactive learn process and thus can be understood faster. Additionally a documentation in a web browser can be displayed for further guidance.

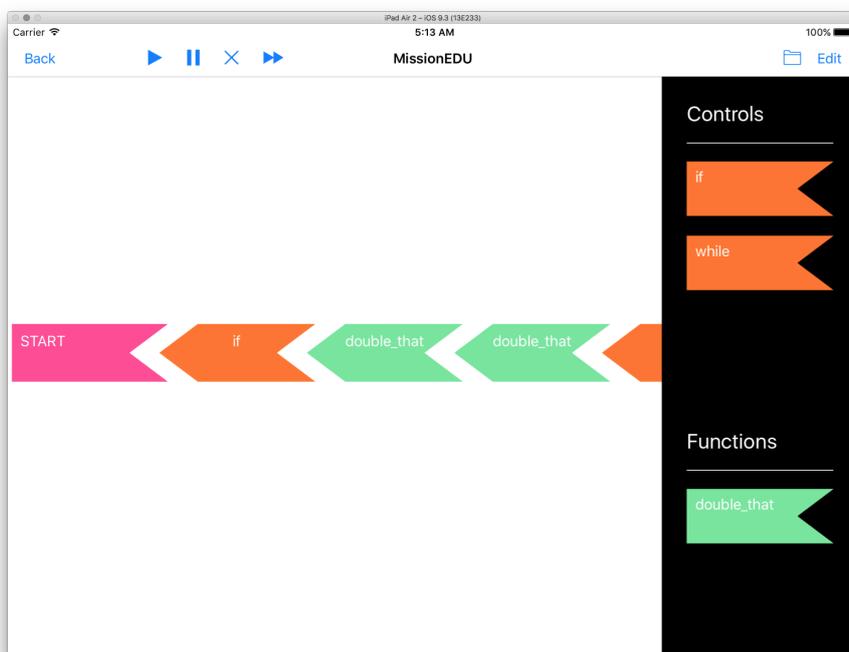


Figure 9.4: MissionEDU Programming Interface

Chapter 10

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.

10.1 Applications of MissionEDU

10.1.1 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.

10.1.2 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.

10.1.3 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.

10.1.4 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. Economic advantages can also be gained, since MissionEDU allows the replacement of defunct hardware as needed.

10.2 Outlook

In the future support for more programming languages for the ARS should be added to further increase the number of supported devices. Especially minding schools - in which there is often a shortage in the budget - the target has to be to support them. Limited resources should be broadened and investments should be made future-proof through the application of MissionEDU in education. The longevity of products should be one of the main goals of MissionEDU.

Furthermore the field of educational robotics should focus on increasing interoperability of their systems. Common standards for software and hardware should be created to enable even students to participate in these events. One way of achieving this would be to take the concept of the MissionEDU Abstract Robotics System and apply it in large scale. With one common industry standard for educational robotics software and robotics systems as well as a unified programming interface the the purview of educational programmes can be broadened and the longevity of learned skills can be increased.

Additionally researchers of the fields of robotics, programming language design, linguistics, human-software-interface design and education should focus on the discipline of graphical programming (GP). With the rise of GP especially in home and workflow automation there is need to develop common concepts for the 21st century.

In the same way MissionEDU should be analyzed in cooperation with educationalists and roboticists to gather information about the impact of such a system on the learning behavior of students. Such research could also be used to improve and increase the longevity of MateScript.

10.2.1 Release

MissionEDU will be released under the MIT-License on <http://github.com/MissionEDU>.

Additionally a website with information about the project, installation guidelines, and student material will be created under <http://missionedu.org>.

Acknowledgement

MissionEDU is an open-source project with the intention to provide an easy entry into robotics education. It is and always will be free. Without support from associations like F-WuTS this would not be possible.

The authors would like to thank Dr. Michael Stifter who was the project advisor and was of great help during the work on this project; DI Harald Haberstroh for his advice; the teachers of HTL Wiener Neustadt for accompanying us on our way from freshman to graduates.

Daniel Swoboda:

I would like to thank my parents and grand-parents for their continuing support as well as the many people I met along the way who are of great importance to me.

Markus Pinter:

As the most important people in my life I would like to thank my parents, grand-parents and all the people who helped me along the way.

Bibliography

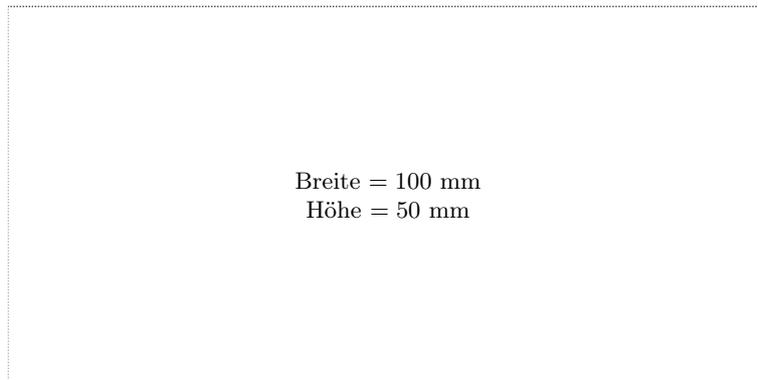
- [1] R. Skidelsky. “Rise of the robots: what will the future of work look like?” In: *The Guardian* (2013).
- [2] Michael Downes Marat Boshernitsan. *Visual Programming Languages: A Survey*. Report. University of California in Berkeley, 2004.
- [3] M. Najork. “Visual Programming in 3D”. In: *Dr. Dobbs Journal* 20.12 (1995).
- [4] D. C. Smith. “PYGMALION: A Creative Programming Environment”. MA thesis. Stanford University, 1975.
- [5] Jiunwei Chen Qinwei Zhu. *Visual Programming*. Online Presentation.
- [6] S Chang. “Visual languages: A tutorial and survey.” In: *IEEE Software* 4.1 (1987).
- [7] M. M. Burnett and M. J. Baker. “A classification system for visual programming languages”. In: *J. Visual Languages and Computing* (1994), pp. 287–300.
- [8] W. Finzer and L. Gould. “Programming by Rehearsal.” In: *BYTE* 9.6 (1984).
- [9] John Maloney; Mitchel Resnick; Natalie Rusk; Brian Silverman; Evelyn Eastmond. “The Scratch Programming Language and Environment”. In: *ACM Transactions on Computing Education* 10 (Nov. 2010).
- [10] Deepak Kumar. “Digital playgrounds for early computing education”. In: *ACM Inroads* 5.1 (Mar. 2014), pp. 20–21.
- [11] GCER. *Global Conference for Educational Robotics*. URL: <http://new.gcer.net/> (visited on 10/23/2016).
- [12] *International Conference on Robotics in Education*. URL: <http://rie2016.info/> (visited on 10/23/2016).
- [13] *Botball® Educational Robotics Program*. URL: <http://www.botball.org/> (visited on 10/23/2016).
- [14] Y. Kuniyoshi I. Noda E. Osawa H. Kitano M. Asada. “RoboCup: The Robot World Cup Initiative”. In: *ICMAS* 82 (1996).
- [15] *The Value of Project Management*. PMI. 2010.
- [16] kanbantool.com. *Kanban: What is it?* Digital.
- [17] Atlassian. *Kanban*. (Visited on 03/30/2017).
- [18] PRIA. *SCORE!* URL: <https://pria.at/research/score/>.
- [19] KIPR. *Wallaby Product Page*. URL: <https://www.kipr.org/hardware-software>.
- [20] P. P. Parikh; M. G. Kanabar; T. S. Sidhu. “Opportunities and challenges of wireless communication technologies for smart grid applications”. In: (2010).
- [21] N. Nurseitov; M. Paulson; R. Reynolds; C. Izurieta. *Comparison of JSON and XML Data Interchange Formats: A Case Study*. Tech. rep. Montana State University, 2009.

- [22] Network Working Group; S. Josefsson. *RFC4648*. Online. Oct. 2006.
- [23] Gerard J. Holzmann. *Design and Validation of Computer Protocols*. Prentice Hall, 1991.
- [24] Stefan Rührup. *Network Protocol Design and Evaluation*. Online Presentation. 2009.
- [25] Stefan Rührup. *Network Protocol Design and Evaluation 2*. Online Presentation. 2009.
- [26] Stefan Rührup. *Network Protocol Design and Evaluation 3*. Online Presentation. 2009.
- [27] Stefan Rührup. *Network Protocol Design and Evaluation 4*. Online Presentation. 2009.
- [28] *What is SPIN? General Description*. URL: <http://spinroot.com/spin/what.html>.
- [29] G. Halfacree. *Raspberry Pi Through History*. URL: <https://www.flickr.com/photos/120586634@N05/>.
- [30] Raspberry Pi Foundation. *Education - Raspberry Pi*. URL: <https://www.raspberrypi.org/education/>.
- [31] Raspberry Pi Foundation. *Software Downloads - Raspberry Pi*. URL: <https://www.raspberrypi.org/downloads/>.
- [32] ELinux. *Raspberry Pi Expansion Boards*. URL: http://elinux.org/RPi_Expansion_Boards.
- [33] Raspberry Pi Foundation. *RASPBERRY PI 3 MODEL B*. URL: <https://www.raspberrypi.org/products/raspberry-pi-3-model-b/>.
- [34] C. Stein. “Botball: Autonomous students engineering autonomous robots”. In: (2002).
- [35] Katalin Popovici and Ahmed Jerraya. *Hardware-dependant Software - Principles and Practice (Chapter 4)*. Springer, 2009.
- [36] NASA. *NASA Computer Room 7090*. URL: <https://commons.wikimedia.org/wiki/File:NASAComputerRoom7090.NARA.jpg>.
- [37] Gilbert Held. *Server Management*. CRC Press, 2000.
- [38] Hermann Härtig Michael Hohmuth Jochen Liedtke Sebastian Schönberg Jean Walter. “The Performance of μ -Kernel-Based Systems”. In: *16th ACM Symposium on Operating Systems Principles*.
- [39] G. Biggs; B. MacDonald. *A Survey of Robot Programming Systems*. Tech. rep. Department of Electrical and Electronic Engineering, University of Auckland, 2003.
- [40] Morgan Quigley Brian Gerkey Ken Conley Josh Faust Tully Foote. “ROS: an open-source Robot Operating System”. In: 2009.
- [41] J. Vlissides; R. Helm; R. Johnson; E. Gamma. *Design Patterns. Elements of Reusable Object-Oriented Software*. Pentice Hall, 1994.
- [42] IST PSU. “Python Programming Language”. In: (2010).
- [43] Python Foundation. *Python 2.7 Release*. URL: <https://www.python.org/download/releases/2.7/>.
- [44] A. A. Porter; H. P. Siy; C. A. Toman; L. G. Votta. “An experiment to assess the cost-benefits of code inspections in large scale software development”. In: *IEEE Transactions on Software Engineering* 23.6 (June 1997), pp. 329–346.

-
- [45] Justin Thiel. “An Overview of Software Performance Analysis Tools and Techniques: From GProf to DTrace”. In: (2006).
 - [46] *ps Linux man page*.
 - [47] Juergen Wolf. *C++ - Das Umfassende Handbuch (translated by the authors)*. Rheinwerk Computing, 2014.
 - [48] *History of Qt*. URL: https://wiki.qt.io/Qt_History.
 - [49] George Basalla. *The Evolution of Technology*. Cambridge University Press, 1988.
 - [50] Marc J. Rochkind. *Advanced Unix Programming*. 2nd ed. Pearson, 2004.
 - [51] C. A. R. Hoare. *Hints on Programming Language Design*. NTIS, 1973.
 - [52] Stephen G. Kochan. *Programming in C*. Sams Publishing, 2005.
 - [53] Adam Freeman. *Windows Presentation Foundation*. Springer, 2010.
 - [54] *Mono Project*. URL: mono-project.com.
 - [55] *Cocoa API*. URL: <https://developer.apple.com/library/content/documentation/MacOSX/>.
 - [56] *Qt API Documentation*. URL: <http://doc.qt.io>.
 - [57] *Cordova API Documentation*. URL: <https://cordova.apache.org/docs/en/latest/>.
 - [58] *Ionic API Documentation*. URL: <http://ionicframework.com/docs/>.
 - [59] Can Bilgin. *Mastering Cross-Platform Development with Xamarin*. Packt Publishing, 2016.

Messbox zur Druckkontrolle

— Druckgröße kontrollieren! —



— Diese Seite nach dem Druck entfernen! —