Extending Empirical Modelling with a physical computing framework

Tapani Toivonen

Master’s Thesis

School of Computing
Computer Science
May 2016
Abstract:
Empirical Modelling (EM) a computer-based modelling paradigm, that was originally introduced in the early 1980s by Meurig Beynon (Department of Computer Science, the University of Warwick, UK). EM aims to offer tools to create artefacts (models) based on the subjective experience rather than accurate scientific observation. Created artefacts rely on three main concepts of EM: observables, dependencies and agents. The early research of EM led to the development of modelling language Eden (Evaluator for Definitive Notations). The current interpreter of Eden is written completely with JavaScript (known as JS-Eden) as is the environment for building the artefacts. Multiple extensions (plug-ins) have been written for JS-Eden [21, 22]. The increasing interest in robotics (especially in Arduinos) created a demand to implement a plug-in for JS-Eden, which would handle the communication between JS-Eden and two popular robotics platforms: Arduino and EV3 (Lego Mindstorms). The thesis concentrates on introducing the implemented plug-in for that purpose. The plug-in currently provides a stable communication channel between Arduino Uno board and EV3 brick as well as a possibility to simulate robot movement in 3D-space without an actual physical robot and a possibility to integrate Q-Learning agent to control the robot (Arduino, EV3 or the simulator robot).

Keywords: Empirical Modelling, radical empiricism, robotics, Arduino, EV3

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EM</td>
<td>Emprical Modelling; computer-based modelling paradigm</td>
</tr>
<tr>
<td>The Construit!</td>
<td>3 year lasting project which aims to develop open learning resources with the tools of Empirical Modelling</td>
</tr>
<tr>
<td>Arduino</td>
<td>A popular robotics and physical computing platform</td>
</tr>
<tr>
<td>EV3</td>
<td>An educational robotics platform provided by Lego</td>
</tr>
<tr>
<td>Q-learning</td>
<td>A type of reinforcement learning, which is used in robotics</td>
</tr>
<tr>
<td>NodeJS</td>
<td>JavaScript runtime environment, which is used for server-side and desktop applications</td>
</tr>
<tr>
<td>JS-Eden</td>
<td>The current platform for creating EM artefacts. Runs on the top NodeJS</td>
</tr>
</tbody>
</table>
## Table of contents

1 Introduction......................................................................................................................1
   1.1 Problem definition.................................................................................................3
   1.2 Research methodology.........................................................................................3
2 Background.....................................................................................................................6
   2.1 Empirical Modelling..............................................................................................6
   2.2 The Construit!.........................................................................................................7
   2.3 The Empirical Modelling tools and Eden language..............................................10
   2.4 Current Empirical Modelling platform: JS-Eden and its extensions.....................13
3 Technical outline of the physical computing framework........................................18
   3.1 Requirements of the physical computing framework........................................20
   3.2 Architecture of the physical computing framework...........................................21
4 Example Applications of physical computing framework.......................................31
   4.1 Example 1: Arduinola............................................................................................31
     4.1.1 Preparations of the SciFest 2016 workshop..................................................31
     4.1.2 Course of events............................................................................................33
     4.1.3 Lessons learnt from the Arduinola workshop..............................................35
   4.2 Example 2: Physical computing framework and Q-Learning............................36
     4.2.1 Machine learning and reinforcement learning..............................................36
     4.2.2 Test case: EV3 robot and the reinforcement learning module.......................40
     4.2.3 Test case results............................................................................................44
5 Discussion and future work.........................................................................................46
   5.1 Lessons learnt.......................................................................................................46
   5.2 Advantages and limitations of the physical computing framework.....................47
   5.3 Future work..........................................................................................................48
6 Conclusions................................................................................................................51
References.......................................................................................................................53
1 Introduction

Empirical Modelling (EM) is an approach to computer-based modelling, which aims to produce interactive artefacts which reflect experienced phenomena [27]. The concept of EM was introduced in the early 1980s by Dr. Meurig Beynon from the Department of Computer Science of the University of Warwick (United Kingdom). EM uses a specific language for creating the models called Eden (Evaluator for Definitive Notations). Eden is an interpreted language with a dynamic type system. The current version of Eden interpreter as well as the modelling environment is written with JavaScript and runs on the top of NodeJS runtime environment [23] and is called JS-Eden (JS stands for JavaScript). The key concepts of Eden (as well as EM) are observables (which are the counterpart of variables in the traditional programming), dependencies (the spreadsheet-like relationships between observables), and agents (wholes that can change observables and their dependencies); all of them reflect the common sense way of understanding phenomena in every day lives.

The Construit! -project (The Construit) began in October 2014 and aims to create open learning resources with JS-Eden. The Construit is led by University of Warwick and has in total 6 partners (University of Eastern Finland being one of the partners). The central theme of the Construit is to devise process of making construals, that is also a core technique exploited in EM. JS-Eden enables the learning resources to be implemented in a collaboration between the learner (a high school student for instance) and the teacher. The use of JS-Eden does not require any previous experience in traditional programming and thus the creators of the artefacts do not necessarily have to be software developers. The Construit also aims to develop JS-Eden, the current EM platform for making construals (MCE) further. For instance the Eden syntax and the user interface of JS-Eden have been under the development during The Construit in order to make JS-Eden more accessible for the novice users. JS-Eden can be extended with plug-ins that increase the breadth of making construals. The available plug-ins include for instance speech synthesis and shared construals over the network.
The increasing popularity of physical computing (and especially educational robotics) in school education [1] has created a demand to build a plug-in for JS-Eden. The plug-in would provide a communication channel between JS-Eden and two different robotics platforms: Arduino (Figure 11, page 16) and Lego's EV3 (Figure 12, page 17). Arduino is an open-source physical computing platform and EV3 is the newest version of the Lego Mindstorms educational robotics series, that offers a kit containing both software and hardware to create programmable robots. The main requirements for the JS-Eden plug-in were defined as follows:

- To feed data to the robot from JS-Eden: For instance if a value of an observable has changed in JS-Eden then rotate a motor

- To feed data from the robot to JS-Eden: For instance feed sensor data to JS-Eden from a robot and automatically change the value of an observable

Possibility for simulating robot movements and the additional sensor data was also one of the minor requirements of the plug-in (physical computing framework). The simulator module was added to the physical computing framework after the communication channel of JS-Eden and both robot platforms were tested to be stable. Later the possibility of integrating Q-Learning agent (a type of reinforcement learning, machine learning, used in robotics) was added to the physical computing framework. The main idea behind the machine learning module (as well as the simulation module) was to provide a complete robotics programming framework for educational purposes and extend scope of JS-Eden and making construals activity to cover an extensive set of physical computing applications.

The structure of the thesis is the following. First the thesis will introduce the concepts EM and the Construit! project as well the main tools of EM. After that the thesis will introduce the physical computing framework; the requirements, the architecture and two test cases, one taking place in May 2016 at the SciFest (an annual science festival for the children) and the other testing a machine learning algorithm combined with the EV3 robot. Finally the thesis will discuss about the advantages and disadvantages of the physical computing framework compared to
traditional approach to program robots (Arduino or EV3) and suggest what could be done in the future. The last chapter (Conclusions) will sum up the earlier findings.

1.1 Problem definition

The thesis will attempt to provide the answers to the three research questions that emerge from the implementation and the use of the physical computing framework. The questions are:

RQ1: Is it technically feasible to implement a complete robotics framework for the existing EM platform?

RQ2: What has to be considered in the implementation?

RQ3: What is the benefit of bringing robotics as part of making construals activity?

1.2 Research methodology

My approach to develop, test and study the physical computing framework is based on the design research framework where the research is embedded within the process of design. The design research framework attempts to generate knowledge during the design process that originally is built on the top of the previous knowledge [18]. The nature of the design research is iterative, where the each cycle of the design generates new information about the current settings and the context of the research and the each cycle thus can be used as a basis of the next cycle. During the design research cycles the research artefact (in this case the physical computing framework) is developed, evaluated and the subsequent feedback refines the design further [18].

My purpose is to develop a complete robotics framework for the existing EM platform so that the users with different purposes can create construals that can be
extended to the physical world, act in the physical word or even learn from the physical world.

Each of the modules created for the physical computing framework were developed during a cycle. At the end of the each cycle the module was tested with the construals developed for the module; for instance after the Arduino module of the physical computing framework was completed, the module was tested with a construal that eventually rotated a motor connected to the Arduino board. After the cycle, a better understanding of the technical requirements of the complete robotics framework was gained as well as the deeper understanding of the frame of reference of EM and robotics in educational purposes. The final evaluation and the reflection of the study of the physical computing framework was made after the test case that took a place during the SciFest 2016 in Joensuu, Finland and the test case that tested whether JS-Eden and the physical computing framework could be used with a machine learning algorithm. The design research's iterative process used in the physical computing framework study is shown in Figure 1.

![Design research and the physical computing framework](image)

**Figure 1.** Design research and the physical computing framework [18]
The used research process during the design, implementation and evaluation can be divided into the phases. Besides the implementation of the modules of the physical computing framework, the process consisted of defining the requirements and the research problems, gaining a theoretical and technical understanding about NodeJS, EM, the Construit! and the educational robotics (Arduino and EV3) and the evaluation and reflection about the results of the study. The phases and related research questions are the following:

• Defining the requirements of the physical computing framework (RQ1)
• Gathering a technical and theoretical understanding in EM, JS-Eden and educational robotics (RQ1)
• Design and implementation of the Arduino module of the physical computing framework (RQ1 & RQ2)
• Evaluation of the Arduino module of the physical computing framework (RQ3)
• Design and implementation of the EV3 module of the physical computing framework (RQ1 & RQ2)
• Evaluation of the EV3 module of the physical computing framework (RQ3)
• Design and implementation of the 3D simulation module of the robotics plug-in (RQ1 & RQ2)
• Evaluation of the 3D simulation module of the physical computing framework (RQ3)
• Design and implementation of the Reinforcement learning module of the physical computing framework (RQ1 & RQ2)
• Test case of the Arduino module at SciFest 2016 (RQ3)
• Test case of the reinforcement learning module (RQ3)
• Reflection and evaluation (RQ3)
2 Background

In the background chapter and its sub-chapters the thesis will concentrate on introducing the key concepts of EM as well as the purposes of The Construit! project, which is a part of the EM domain. The background chapter also introduces some previous variants of EM environments, the syntax of Eden and the current EM environment, JS-Eden.

2.1 Empirical Modelling

The philosophical foundations of EM have been built on the top of radical empiricism: a philosophical doctrine introduced by an American philosopher William James (1842 – 1910) [16, 20]. Radical empiricism stresses the value of the relationships between the subjective experiences by which we associate one element with another where the traditional empiricism argues that we purely separate objects by their relationships. Thus one of the key concepts of EM are dependencies [3]; the relationships between observables. The traditional computing tends to have an underlying presumption that all computer-based modelling can be accounted for by using the universal abstractions and thus disregard the importance of a subjective experience in the modelling [3]. EM artefacts (construals) all share three key concepts: observables, dependencies and agents. The construal can be seen as a combination of these three concepts, which are the characteristics of a phenomena, which is observed. A major feature of EM is the use of definitions as means of representing state-as-experienced [11]. State-as-experienced is a term used to describe that nature of any built construal. Where the state of a classical computer-based model in constructed in a basis of scientific observation (referent) EM philosophy argues that the association between a construal and its referent is in contrast more subjective and personal and the association takes the form of a directly sensed relationship between the experience of the construal and experience of the referent of the construal [13]. The activity of making construals is visualized in Figure 2.
Definitions are the simple way in EM to represent how the value of one observable is dependent on the value of other observables. Definitions are formulated using definitive notations, that represent the language, which is used to create definitive scripts (construals) - the set of definitive notations. Definitive notations in the context of EM are used to address scalar, textual and list data, line drawing and 3D models [13]. The language used in EM to create construals is called Eden. Eden language has been under the development since the year 1987. Currently Eden has two web-based variants Web-Eden (2008) and JS-Eden (2011). Other variants of Eden include tkeden (implemented using tk/tcl interface) and Cadence (2009), which offers a support for the transition from construals to classical computer programs [12].

2.2 The Construit!

Making construals is argued to be a way of using computer to help us to better understand a phenomenon [5]. Making construals is a core theme in a three year EU Erasmus+ -project The Construit!. The project develops open educational resources in a collaboration between educators and students with the EM, JS-Eden being the most important environment for making construals (MCE). The Construit! is led by
the University of Warwick and the key partners are from Finland, Greece, Slovakia, Netherlands and UK. The Construit! as a subset of EM is shown in Figure 3.

![Diagram](image)

Figure 3. The Construit! as a subset of EM [4]

Unlike a traditional computer program, making construals establishes "an open-ended context for interaction within which the human and automated agency in a domain can be expressed [5]". The open-ended approach mixes the educators and the students to work in the same environment (JS Eden) in a way that both the students and the educators can interact with the same teaching artefact (construal). Existing construals for the educational purposes are for instance a model of sin-wave and a model of the value of coins. Sin-wave construal can be seen in Figure 4.
In contrast to traditional computer program, which tends to be a formal recipe for an action that has abstract functional specifications, concrete implementation and user-interface, construal attempts to provide interactive experience to the maker of such artefact. While traditional programming focuses on writing programs as a solution to a problem, EM tends to put its primary emphasis on making construals which are created incrementally and which take states and transitions closely connected with an object of study [17].

It is argued [17] that by integrating the learner and the educators in the same open-ended environment provides a potential in learning disciplines:

- The learners can create learning resources and thus adapt the roles of educators in some degree.
- The both educators and learners can modify the existing open learning resources.

Although creating open educational resources (OERs) is valued by the educators, the practical uses of the OERs tend to be quite limited because the modifying the resources may need the skill of a developer. Making construals is giving the educators much more accessible environment; if an educator wants to modify the existing EM artefact for the purposes of his or hers own, it is possible to modify for
instance the values of some core observables in the JS-Eden after identifying them first. The update of the construal takes place in the same environment as the written notations.

Despite the differences between in making construals and traditional programming, it has been argued that making construals requires some understanding in for instance iteration (for loops and while loops), conditions (if, else if, else -structures), functions and procedures [4]. Despite the similarities in language syntax and expressions between in making construals and traditional programming, making construals is said to lower the gap between the computing professionals [4].

2.3 The Empirical Modelling tools and Eden language

During the history of EM, various tools for building EM artefacts have been developed. The current version of Eden interpreter is completely written in JavaScript and runs in a web browser. The development of JS-Eden began in 2011. JS- Eden is the second web-based EM environment, Web-Eden (developed in 2008) being the first. An attempt to build a more accessible environment was the primary reason for Eden to be a "web-enabled" language. JS-Eden is introduces in a more specific manner in the next subsection.

Web-Eden differs from JS-Eden in a sense that Web-Eden interface uses Flash where JS-Eden is written in JavaScript and HTML5. Web-Eden instances run on a top of Microsoft Server whereas JS-Eden instances run on a top of NodeJS. Besides tools of Web based development, Java programming language has been used to develop some of the versions of Eden; such implementations include jeden, JaM and MoDD. One of the earliest (desktop) versions of Eden was written with a tk/tlc library to create a graphical user-interface [13, 14].

The syntax of Eden language has developed over time but still the main concepts in the language have been left unmodified: observables, agents and dependencies. The current syntax of Eden language allows user to use basic arithmetic operations (+, -, /, *, \\%) as well as the basic value assigning where the identifier is on the left hand side of = -sign and the value is on the right hand side. The statements
are terminated with a semicolons. The syntax of Eden also allows the use of conditions (if, else if, else) and loops (for-loops and while-loops as well as the do-while-loops). The syntax of ternary condition was first adopted from C-like languages (identifier assignment-sign condition ? Value-in-case-condition-is-true : value-in-case-condition-is-false) but in the latest version of Eden the ternary condition is more Python-like (to improve the readability): identifier assignment-sign value-in-case-condition-is-true condition else value-in-case-condition-is-false:

```
someOberavbles = someValue if condition == true else someOtherValue;
```

The conditions in Eden use the C-like comparison operations (==, <=, >=, !=) [13, 9,]. There are no naming conventions in Eden; the user can use lower and upper case alphanumeric characters and underscores in the names of identifiers (observables and functions for instance).

The blocks of Eden languages are represented with curly brackets - in a C-like manner. Blocks are used to define loop-blocks, conditional blocks and for instance function blocks. Although the indents in the construals can be used to clarify the notations, special indents are not needed by the interpreter. The functions are defined as shown in Figure 5.

```
func identifier {
    parameterOne, parameterTwo;
    function body
    return someObservable;
}
```

**Figure 5. Example of the function in Eden**

Functions in Eden always return the value of an observable. If the return statement is omitted, the function will return undefined. Besides the funtions, Eden allows user to write procedures. Procedures in Eden do not return any value and are triggered by the change of an observable. The procedures are written as shown in Figure 6.
Furthermore Eden has a dynamic type system. Dynamic type system means that the type of – in the case of the Eden – observable is verified during the program's runtime; thus user does not define the type of the observable explicitly during the creating construals.

Unlike traditional programming languages, Eden lacks variables and replaces the variables with observables. The observables can be strings, floating numbers, integers or arrays for instance. Observables can have a value of their own or the value of an observable can be dependant on the value of another observable(s). The dependency between two (or more) observables is represented in a way that by changing the value of the observable, the value(s) of observable(s) that are dependent on that observable is also updated automatically to correspond the dependency. The dependency between two observables is created by using the "is" keyword:

```plaintext
myObs = 1;
myObs2 is myObs * -1;
```

The value of myObs2 is now -1 but if the value of myObs is changed for instance to -2, the value of myObs2 is automatically updated by the Eden interpreter to 2. The agency in Eden can be understood so that the agent is a set of notations, which ultimately can change the values of observables and ultimately can consists of the observables, which are only available in the agent's scope. The following set of statements present an example of an agent as shown in Figure 7.
func exampleAgent {
    para onlyPara;

    auto obsInAgentScope; ## local observable
    obsInAgentScope = "I am a string";
    onlyPara = obsInAgentScope // " " // onlyPara;
    return onlyPara;
}

Figure 7. Example of an agent in Eden

Where two hashtags are one line comment and // is for joining strings together. Thus for example functions in Eden can be understood as agents.

The current version of Eden provides a special canvas picture, which is used to create a graphical presentation of the artefact. Some examples of built-in functions are Circle and Rectangle for drawing 2D graphics. Eden also provides libraries for the generating random numbers and calculating mathematical functions.

2.4 Current Empirical Modelling platform: JS-Eden and its extensions

JS-Eden is a JavaScript based variant of the Eden interpreter. The development of JS-Eden began in 2011. JS-Eden is a open source project available on GitHub [23]. The newest version from the master branch of JS-Eden is hosted by the University of Warwick but JS-Eden can also be run locally as long as the user has NodeJS installed. The architecture of JS-Eden instance while running is shown in Figure 8. JS-Eden runs with Google Chrome and Mozilla FireFox browsers and the current version cannot be used with Internet Explorer or Safari. JS-Eden was originally developed by Tim Monks as a part of his MSc thesis in 2010 - 2011. Nick Pope has been the principal developer of the JS-Eden but important contribution have also been made by Elizabeth Hudnott and Jonathan Foss [10]. JS-Eden works as a
prototyping environment for Making Construals Activity. The current user-interface of JS-Eden provides a special input window to write construals, which is named "Script View". The visual state of the construals is shown in a window named "Canvas Picture". The user interface of JS-Eden is when loaded to browser is shown in Figure 9.

Figure 8. The architecture of JS-Eden instance running
Instead of executing the whole construal, a user can control the execution flow by executing the definitions one by one in whatever order the user wants to. It is also possible to execute the whole construal so that the execution flow is linear. JS-Eden has a real-time syntax validator, which will inform the user if the construal has any syntax errors. After the execution of the definitions, the user can automatically see the changes in the observable values from the window named "Observable List" as shown in Figure 10. Also the red rectangle's dependency on the observable is shown in Figure 10.

JS-Eden ships with a number of pre-written construals for demonstration and for educational purposes. The pre-written construals can be modified by anyone following the open-ended philosophy of EM. The pre-written examples include "sinewave" to model a mathematical phenomenon of sine wave, "logo" to model the logo programming language and "givingchange" to model the values of coins. Besides the three examples given, there are over 50 other construals available in JS-Eden [23]. JS-Eden supports version controlling and anyone can log in to JS-Eden instance hosted by the University of Warwick with their Google account to post the their own construals (modified construals or a construal written from the scratch).
The current syntax of JS-Eden allows users to write definition blocks, which are executed whenever the condition of a "when-clause" is true:

```
when (someCondition) { ... }
```

Another nature of the syntax of JS-Eden are blocks which define actions. Actions can be understood as macros without a return value and when-clauses as procedures which are triggered on some event. The actions are written as follows:

```
action actionIdentifier { ... }
```

and executed:

```
do actionIdentifier;
```

During the development of JS-Eden, several extensions (plug-ins) have been developed for JS-Eden. Plug-ins are written in JavaScript and can be run either locally or in the browser. The extensions are included and loaded from the user-interface of JS-Eden. Extensions include for instance speech synthesis, observable mining and network remote, which allows multiple users to modify the same construal at the same time. The network remote extension consists of the client-side script, which allows user to connect to a remotely hosted server-side script. All instances connected to the server-side script share the same Script View and the canvas picture enabling real-time collaboration between users. The network remote extension, which uses NodeJS libraries such as WebSocketServer served as a model for building the robotics extension.
Figure 10. Canvas picture, Script View and Observable list in JS-Eden. Observable `myOtherObs` underlined
3 Technical outline of the physical computing framework

The increasing interest in certain robotics platforms (Arduino, Figure 11, and EV3, Figure 12) among the members of The Construit! stimulated the development of physical computing support of JS-Eden. The first prototype of such a plug-in was demonstrated in December 2015. The first prototype used a modified Network Remote extension so that observable values were sent to the Network remote server socket from the Network remote client and the server passed modified data via the serial port to the Arduino board. With the first prototype, it was possible to turn a servo motor and light up a led light from JS-Eden. However the communication took place only in one direction: from JS-Eden to Arduino. Also the primary target was for JS-Eden to communicate with Arduino with a USB cable. The extension was developed further this thesis. In this chapter we will discuss the development of the extension. First this chapter will discuss the requirements of the extension. Then the chapter will introduce the architecture of the extension and finally the chapter will discuss the possible advantages and disadvantages of the physical computing framework compared to other Arduino and EV3 remote-control programmable interfaces.
Figure 11. Arduino board and sensors connected to it

Figure 12. EV3 brick and gyro sensor
3.1 Requirements of the physical computing framework

Major requirements of the physical computing framework were identified in December 2015 when the The Construit! meeting took place in Warwick [8]. The first requirement was to establish a communication channel from JS-Eden to Arduino. The first prototyped version (December 2015, not written by the author of the thesis) was a modified version of Network Remote plug-in that passed observable values via Net-work remote client to Network Remote (NR) server and via NR server to Arduino. Network Remove plug-in however does not listen to all observable changes; only observable changes that are manually executed by the user are sent to NR Server (and in the prototype further to Arduino). A part of the requirement that the JS-Eden could pass data to a robot was that all observable changes had to be passed from JS-Eden to a robot.

The second requirement was that a robot could also pass data (for instance sensor data) to JS-Eden and JS-Eden would update a corresponding observable value as defined with a dependency. An example could be a situation, where an observable is dependent on a ultrasonic sensor value (the distance between a sensor and a physical object in the front of the sensor) and the sensor value changes, the observable value also has to be changed automatically.

Furthermore the communication channel between the robot and the JS-Eden was supposed to operate wirelessly. Some alternatives were considered first (such as WiFi) but Bluetooth was chosen instead. Although the physical computing framework was supposed to enable communication wirelessly via Bluetooth, USB-serial port communication was also one of the requirements (in case for instance the Arduino board had no Blueooth dongle). The prototype (December 2015) used a custom Arduino-script (ino-file), which read buffer data from the serial port and then based on the buffer data, executed commands from JS-Eden. During the planning of the final physical computing framework, Firmata firmware with Bluetooth support was considered at first but it came soon clear that Firmata firmware did not provide a stable Bluetooth connection, so a custom ino-file was written to listen to serial port data in Arduino. Firmata is an Arduino protocol, which enables remote control of the
At first (December 2015) there were no plans to enable the communication between JS-Eden and EV3 robots, but the support was added to the physical computing framework in March 2016 in order allow the users to decide between several robot platform options while using the physical computing framework. The development of the physical computing framework was initially started in January 2016. EV3 brick’s firmware directly supports Bluetooth communication with (for instance) a computer so there was no need to replace EV3 brick’s firmware. The communication protocol (for instance for rotating motors) is available online [15].

The EV3 support in the physical computing framework was intended to enable the same features that the Arduino part of the physical computing framework did: pass sensor data to JS-Eden and update the bound observable values automatically and pass JS-Eden data to EV3 so that for instance a motor rotates. The requirements of the physical computing framework did not specify any needs for the 3D simulation of a robot nor any needs for Q-Learning integration. However both were added to the physical computing framework in March / April 2016 in order to create an aesthetically more attractive and technically more complete robotics environment within JS-Eden; there has previously been some debate on the visual attractiveness of the artefacts written in Eden [25]. With 3D simulation the user can experiment with constructions written to control robots in more realistic environment.

3.2 Architecture of the physical computing framework

The physical computing framework consists of five modules (all written in JavaScript), that are simultaneously loaded when the user loads the physical computing framework from the Plug-in manager via the JS-Eden user interface. The modules are:

- Robotics client: Loaded to web browser from JS-Eden, listens to observable changes and passes JSON data further to (physical computing framework) core modules or accepts JSON data from the core modules and updates values of the
observables. The data passing and receiving to and from core modules is based on the prefixes of observable names.

- Arduino server: A web socket server that receives JSON data from the robotics client and generates a data buffer, which is sent to a serial port (connected to Arduino board). It also receives a buffer data from the serial port and passes generated JSON data to the robotics client.

- EV3 server: The same as Arduino server but communicates with an EV3 robot

- Robotics simulator: Uses WebGL to render a 3D environment to simulate a robot and the surrounding environment of the robot.

- Reinforcement agent: A Q-learning agent (uses a recurrent neural network to store Q-values). Can be used for instance to control the robot.

The only module that has a user interface is the robotics client that is loaded to the browser as a window from the JS-Eden. The client listens to all observable changes and filters those observables that have no suitable prefix in the name of the observable. After the suitable observable has changed, the client passes JSON object to the one of the modules. JS-Eden has no public API, so some reverse engineering had to be done in order to write a listening JavaScript function for the client. The architecture of the physical computing framework and its relationship between the Eden can seen in Figure 13.
If the prefix of the observable name is "arduino_" then the client passes JSON data to a Arduino server. The Arduino server has to be started separately with NodeJS. The Arduino server is started with command line arguments to run the server either in Bluetooth mode or in the USB mode. The Arduino server receives JSON data from the client and generates a buffer array that contains the information as to whether JS-Eden wants to read values from a certain Arduino board pin or write value to a certain Arduino board pin. If JS-Eden wants to read a pin value (for example an analog value from the sensor attached to an analog pin), Arduino will write buffer data to a serial port, which is listened to by the Arduino server every time the value has been changed. If JS-Eden wants to write data to an Arduino pin, writing only happens once when the Arduino receives a buffer array from the Arduino Server. Prefixes of the observable names that are accepted by the client are:

```javascript
arduino_digital_identifier = [pinNumber, value];
```
Sensors are assigned to null values (@ in Eden equals null) and updated automatically when the sensor values change. Actuator-observables consist of pin number and pin value pairs, which are presented as arrays. For the actuators, bound observables can also contain dependencies, so for instance:

```java
myValue = 255;
myPin = 6;

arduino_analog_myArduinoPin is [myPin, myValue];
```

would change the pin number and the pin value when the values of myValue and myPin are changed. If the user wants to use a servo motor, the user has to plug a servo to pin number 9 in the Arduino board. The user interface of the Arduino server module of the physical computing framework is shown in Figure 14. When the Arduino server is running and connected to an Arduino board, the user simply click
"Connect" button to establish the connection between JS-Eden and the Arduino server, which is connected to the Arduino board.

The data flow is being visualized in Figures 15 and 16. In order for the Arduino server to communicate properly with the Arduino board, a special ino-file has to be uploaded to the Arduino board first. The ino-file is included with the physical computing framework and can be compiled to be uploaded with Arduino IDE.
Figure 15. Data flow of Arduino module (From JS-Eden to Arduino)
The EV3 part replicates the data flow of Arduino part of the physical computing framework. The EV3 server establishes the Bluetooth connection with EV3 brick and the client connects to the web socket of the EV3 server to communicate with the EV3 brick. Sensors that are supported by the EV3 module are infrared, gyroscope, ultrasonic, touch and color sensors. They are used by using the prefixes:
The motors are used with the array containing port value pairs:

```javascript
ev3_motors_identifier = [motorChar, motorValue, otherMotorChar, otherMotorValue];
```

In case a user wants to experiment with the built model for instance a model used to control robot so that the robot moves around in the room but avoids any obstacles, the user can model the environment in 3D space with a simulator module of the physical computing framework. The simulator module automatically loads an obj-file (a geometry definition file adopted by 3D graphics application vendors) and creates a robot-like object to the simulator window. User can then build a 3D environment where the user can control the "motors" of the robot-like object. The user also has a change to bind pre-defined observables to a robot-like object’s position in 3D space as well as the rotation of the robot-like object. The simulator module is written in a top of three.js (a WebGL library for JavaScript) [26] and is loaded automatically with the physical computing framework. The user can add the shapes such as cubes,
spheres, cylinders, curves and rings. The added objects act as regular observables and thus support fully any dependencies. 3D was chosen to the simulator to make the simulator window more visually attractive and powerful and to better simulate the movement of the robot. An example of built construal and the simulator robot are shown in Figures 18 and 19.

Figure 17. EV3 user interface of the physical computing framework
**Figure 18.** The simulator module of the physical computing framework and construal to control robot’s motors

```
simulator_cube myCub is [405, 40, 40, 0, 0, 227, 0, 0, 0, "green"];
simulator_cube myCub2 is [405, 40, 40, 193, 0, -10, 0, 11, 0, "yellow"];
simulator_robot_motors is ["left", 0, "right", 0];
```

**Figure 19.** The simulator module and the motors observable triggered to drive the robot forward

```
simulator_cube myCub is [405, 40, 40, 0, 0, 227, 0, 0, 0, "green"];
simulator_cube myCub2 is [405, 40, 40, 193, 0, -10, 0, 11, 0, "yellow"];
simulator_robot_motors is ["left", 30, "right", 30];
```
In this chapter, the thesis will introduce two different applications that make use of the different modules of the physical computing framework. First, the thesis will introduce the Arduinola application that was developed for the SciFest 2016. After the Arduinola application, the chapter will introduce an application that uses machine learning implementation with the EV3 module of the physical computing framework. The EV3 robot gathers data from the environment and passes the data to the machine learning agent that tries to learn how to avoid obstacles. The evaluation of two applications was based on the technical features of the modules being tested. In detail, features such as the stability of the communication channel between the server and the robot platform and the updates of the observable values bound to module entities (such as motors) were observed.

4.1 Example 1: Arduinola

SciFest is an annual science festival for the school children arranged by Joensuu Science Society. SciFest is a three days long festival and it takes place in Joensuu Areena. SciFest features a variety of workshops from different organizations such as the University of Eastern Finland, Joensuu Science Society, Carelia University of Applied Sciences and some private companies. The Construit project organized two different workshops at SciFest. The Arduinola workshop was so called non-stop workshop where children could attend any time they wanted. The Arduino module and the simulator module were both tested during the workshop.

4.1.1 Preparations of the SciFest 2016 workshop

The idea of the workshop was to present the Arduino module of the physical computing framework. The members of the Construit came up with the idea of Arduinola. Arduinolas would be ball-like creatures that sense light, sound and temperature for instance. Some of Arduinolas would enjoy bright lights and some of Arduinolas would enjoy loud noises. The mission of the children attending to the
workshop would have a change to move around Arduinolas to the places in the workshop to make them happy. The happiness of Arduinola would be seen from the JS-Eden so that JS-Eden would draw the face of Arduinola and if Arduinola was in the correct place, Arduinola would smile in the JS-Eden or if Arduinola was not in the correct place (there would be too much light for example), Arduinola would not be smiling. Due the open nature of construals, the children attending to the workshop would also have the possibility to edit the code and to change the values that make Arduinola happy or even to change the looks of Arduinola. The introduction of the Arduinola workshop from the SciFest website (http://www.scifest.fi) is:

Arduinolas are cube-like creatures which react to sounds, light and which way they are facing. Like us, they are all different. Some of us like loud noises and bright lights. Others like to lie quiet on their back in the hot sun. In this workshop, you will get a chance to experiment with some Arduinolas and find out what they like. See if you can put baby Arduinola to sleep, get his big brother out of bed, and make his little sister eat broccoli. When you've done this, take a visit to the Arduinola laboratory, where you can find out what makes an Arduinola tick. If you like, you can even make your own Arduinola, and see if your friends can figure out what it likes best.

The physical appearance of Arduinola was created with the chicken net, paper mass and acrylic colors and the Arduino board with Bluetooth module, sensors and the battery were inside of the Arduinola. Arduino board passed sensor data to the Arduino server module and Arduino server module passed the JSON data to the robotics client that updated the looks of Arduinola that was shown in the robotics simulator module. I chose the 3D simulator over the traditional canvas picture in order to create more appealing looks of Arduinola for the children (Figure 19).

In total four Arduinolas were created using Arduino Uno or Mega. Four Bluetooth modules were used with heat, humidity and light sensor. Workshop area was equipped with hair dryer and a bright light to generate heat and light. In this way, children could change Arduinola's environment. JS-Eden was hosted in four instances of Ubuntu Linux laptops.
4.1.2 Course of events

During the first day of Arduinola workshop three Arduinolas were assembled and connected to three laptops. Each laptop run JS-Eden with the physical computing framework and each JS-Eden instance run the same Arduinola construal. Arduinola construal had two observables bound to two sensors (heat and humidity sensor and lighting sensor). The looks of Arduinola was created with curves and spheres from the simulator module (Figure 20). The construal used to create Arduinola is shown in Figure 21.

![Figure 20. Arduinola from the simulator window, SciFest 2016](image)
simulator_arduinola_ard is [27];

## bind observable to Arduino's analog pin 3 and 5:

arduino_analog_sensor_5 = @;

arduino_analog_sensor_3 = @;

## bind observable to simulator's curve 3D object:

simulator_curve_antenna is [ -5, 183, 4, -28, 596, 0, 150, antennaHappy, 0, "yellow" ];

simulator_sphere_eye1 is [ 50, 50, 50, 108, 186, -119, 50, 59, 50, "yellow" ];

simulator_sphere_gaze is [ 21, 12, 25, 114, eyeHappy, -171, 50, 50, 50, "white" ];

simulator_sphere_gaze2 is [ 21, 12, 33, -51, eyeHappy, -171, 50, 50, 50, "white" ];

simulator_curve_eyelid1 is [ 163, 256, -166, -109, eyelidHappy, -216, 82, 306, -221, "green" ];

simulator_curve_eyelid2 is [ -4, 256, -166, -39, eyelidHappy, -216, -88, 278, -221, "green" ];

simulator_sphere_eye2 is [ 50, 50, 50, -66, 183, -119, 50, 50, 50, "yellow" ];

simulator_curve_mouth is [ 153, 60, -166, -36, mouthHappy, -216, -137, 68, -173, "yellow" ];

likesLight = 10;

likesHot = 10;

## generate dependencies about happiness:

happyLight is ((10 - abs(likesLight - (arduino_analog_sensor_3 % 10))) * 10);

happyHot is ((10 - abs(likesHot - (arduino_analog_sensor_5 / 2))) * 10);

happy is 250 + happyLight + happyHot;

eyeHappy is 0.5 * happy;

eyelidHappy is 0.8 * happy;

mouthHappy is 550 - 1.1 * happy;

antennaHappy is 100 + happy;arduino_analog_sensor_3 = @;

Figure 21. Construal for Arduinola. Happiness is dependent on the sensor values

The mouth and the eyelids of Arduinolas were dependent of the observables bound to the sensors and when the Arduino board passed the changed values to the JS-Eden,
the looks of the mouth and the eyelids were changed based on the observables that determined if Arduinola was happy in the bright light and warmth. Yet during the first day of SciFest, the paper mass figures were not used because they were still drying from the painting. During the second and the third day only one paper mass figure of Arduinola could be used because the painting damaged other and the paper mass started to fall into pieces.

Arduinola workshop attracted a substantial amount of attendees at SciFest. From time to time, the workshop was even crowded, which led to some issues since there were only two tutors available; the author of the thesis and one other computer science student from the University of Eastern Finland. The visitors were mainly school children but also teachers were visiting. About half of the children were girls. Their age varied from 6 to 13 years. Some children only moved Arduinolas around (for instance moved the lighting sensor to the light and shadow) and some were experimenting with both sensors and the Arduinola construal.

4.1.3 Lessons learnt from the Arduinola workshop

Attention during the workshop focused on the technical observations of the Arduino module and the simulator module and how they worked together. When the children for instance exposed the lighting sensor to the light, the Arduino board passed the data to the Arduino server that passed the formatted data to the robotics client. The robotics client updated both the observables bound to the sensor and the looks of Arduinola (via observables bound to the simulator module's 3D objects). The physical computing framework updated the observable values from the sensors and finally the looks of Arduinola with no exceptions.

However the connection between the Arduino board and the Arduino server was not always stable; the Bluetooth connection between the Arduino board and the Arduino server dropped once per two hours. It did not become clear whether the connection dropped because of the bug in the Arduino server code or because the Bluetooth module in Arduino drained the batteries quite quickly. Thus it is suggested that the Bluetooth module would be used in the Arduino module only if there is no need for a longer communication period.
To summarize, the concept of the Arduinola construal is interesting and working but I suggest that the connection between Arduino board and the Arduino server of the physical computing framework should be established with USB cable because the Bluetooth connection can be unstable.

4.2 Example 2: Physical computing framework and Q-Learning

Another test case for the physical computing framework is an implementation of a machine learning construal. For the implementation I chose Q-learning, which is a type of reinforcement learning algorithm. Q-learning is widely used in the robot applications [24]. Machine learning has been used to control and to simulate robotics for educational purposes in open-source projects such as Simbad, Webots and Player [19]. Simbad is a Java interface that enables user to create 3D environment for the robot and simulate the movements of the robot in such environment. Simbad also allows user to add machine learning implementation to the created environment and the robot. Simbad is designed for the educational purposes for example to teach the basics of the machine learning combined with the robotics with the Java programming language [19]. In order to create a full robotics framework for JS-Eden and increase the possible use scenarios of the framework, the machine learning module was added to the physical computing framework. The idea was to allow user – not only to control a robot but also to define a machine learning agent as Simbad does but in EM style: to use making construals approach to control machine learning in a construal. The machine learning module was tested with an EV3 robot. In the test implementation, a machine learning agent was embedded in the physical computing framework trying to learn from ultrasonic sensor data that was sent from the EV3 module to machine learning agent to avoid obstacles.

4.2.1 Machine learning and reinforcement learning

Machine learning is a field of study in the computer science that intends to develop algorithms that can learn from the data and make predictions from the data. One subfield of the machine learning is so called reinforcement learning. In the reinforcement learning an intelligent agent (the implementation of an algorithm) acts freely in the environment and receives feedback about its current state. Based on the
current state, the agent chooses an action from the set of the pre-defined actions in order to obtain a new state. Based on the choice, the agent is rewarded or punished. The goal of the reinforcement learning algorithm is to maximize the reward gained in a long run. The reinforcement learning algorithms are widely used in the field of autonomous robots for example in the self-driving cars. One of the implementations of the reinforcement learning is called Q-learning. Q-learning can be viewed as a method that provides agent with the capability of learning to act optimally in the given environment; that is to maximize eventually the reward. In Q-learning, the agent stores the rewards (so called Q-values) and the state where the reward (or the punishment) was awarded. Usually, the Q-values are stored in the multidimensional array where the rows and the columns represent the states of the agent. However, the array implementation is not feasible in the large environments (large amount of the states) because the size of the array will increase exponentially. In the large environments, more optimal solution is to store the Q-values in the artificial neural network [6].

Artificial neural networks (NNs) are inspired by the biological neural networks that consist of artificial neurons and the weights between the artificial neurons. The weights between the neurons determine the volume of the signal (input) travelling through the NN; the larger the weight is, more powerful the signal is. Each neuron has an activation function that converts the input of the neuron to the output of the neuron. As in the biological neural networks, the NNs can have different layers that are connected. In the simplest form, the NN has two layers: the input layer and the output layer. Every neuron of the input layer is connected to every neuron of the output layer. Usually, NN has more than two layers; so called hidden layers in the between of the input layer and the output layer [6]. Artificial NN is shown in Figure 22.

The learning process of the network is based on the adjusting of the weights of the network. In the Q-learning, the input layer takes the current state as an input and the output of the output layer is based on the current state. The action is the largest value of the output layer (so called maximum of the arguments). So for example if the actions are ["rotateMotorAForward", "rotateMotorBForward", "rotateMotorABackward", "rotateMotorBBackward"] and the maximum value of the
output layer comes from the neuron one, the action is "rotateMotorBForward". After
the action is given by the NN, the reward is given and the reward is propagated
backwards ("backpropagated") to NN from neuron one of the output layer.
Backpropagation is the process of adjusting the weights of NN starting from the
output layer and ending to the input layer. The goal of the backpropagation is to
adjust the weights so that the input would generate the optimal output.

The implementation of the machine learning module consists of a recurrent neural
network, which serves as storage for the algorithm’s Q values. The recurrent neural
network (RNN) differs from the basic NN (the so called feed forward NN) in a way
the RNN uses its own hidden layer as a part of its input layer. This allows RNN to
have a temporal memory. The hidden layer used as a part of the input layer is the
hidden layer from the previous state of the RNN. Also the learning algorithm of the
RNN differs from the basic backpropagation algorithm used in feed forward NN; the
previous inputs and outputs are stored as an experience and the backpropagation is
applied iteratively (iterations) from the newest state (input and output pair) to the oldest state. The recurrent neural network was chosen over the feed forward network because it is more powerful (due its temporal memory ability) and a neural network implementation over an array implementation for the same reasons. The recurrent neural network in the physical computing framework uses its own hidden layer as a part of the input layer.

The client module of the physical computing framework passes the states to the reinforcement learning part, which passes the chosen action from the action array. The user then updates the reward of the action, which is then passed forward by the client module to the reinforcement learning module. The user can change the learning rate, epsilon greediness (the probability of the random choices of the agent), gamma value (the learning rate of the agent) and the number of hidden units as well as the states and actions of the Q-learning agent. The activation function of the neural network is the hyperbolic tangent and the backpropagation though time function is a derivative of sigmoid. The backpropagation through time is a backpropagation algorithm that takes the previous states of the NN in the consideration. The reinforcement learning module of the physical computing framework can be used to teach the network to control robot (for instance to avoid obstacles). The usage of the reinforcement learning module goes as follows:

• `reinforcement_agent` is [
  
  [states], [actions], numberOfHiddenUnits
]

  : Initializes the reinforcement agent and creates an input layer with as many nodes as the size of the states array. The size of the output layer is the size of the actions array. `numberOfHiddenUnits` is optional and initializes that size of the hidden layer.

• `reinforcement_action` = @;

  `reinforcement_reward` = 0; : Binds the observables to an action and to the reward. Action is automatically updated to correspond to the maximum of the arguments (an element from the actions array above each time the output is generated by the network.

39
reinforcement_learn = false / true; : Triggers the learning or the action: when set to false, the network generates an output and passes the output to the client module of the physical computing framework. The client module then automatically updates reinforcement_action to correspond to the chosen action from the initial state array. The states during the action are the current values of the states array. When set to true, the client module of the physical computing framework passes the current value of the reinforcement_reward to the reinforcement learning module of the physical computing framework and the weights of the network are updated based on the reward and the last action.

As with the simulator module, the reinforcement learning module is also loaded automatically together with the physical computing framework; Arduino and EV3 modules are servers whereas the simulator and reinforcement learning modules work in the browser environment.

4.2.2 Test case: EV3 robot and the reinforcement learning module

For the test case of the reinforcement learning module I built an EV3 robot with two motors for the movement and an ultrasonic sensor for gathering data from the environment. The idea was to connect the EV3 robot to JS-Eden with EV3 module of the physical computing framework and feed the ultrasonic sensor data to the reinforcement learning agent of the reinforcement learning module of the physical computing framework. EV3 robot moved freely in a room with obstacles such as a couch, chairs and walls. The EV3 brick and the EV3 server module of the physical computing framework communicated with the Bluetooth connection. The test case took place in June 2016.

The current value of the ultrasonic sensor passed to JS-Eden was the state of the reinforcement learning agent and the actions were "rotate both motors forward", "rotate both motors backwards", "rotate only left motor forward" and "rotate only right motor forward". The agent was rewarded positively (the reward was a positive integer) if the agent chose forward action when the ultrasonic sensor did not sense any obstacle ahead. On the other hand, the agent was rewarded positively also if it chose to rotate motors backward when the ultrasonic sensor sensed that there was an obstacle ahead. Rotating motors backward when there were not any obstacles or
rotating motors forward when there were obstacles would result in rewarding (or punishing) the agent with a negative reward (negative integer). The goal was to study whether the reinforcement learning agent could learn to drive directly and avoid obstacles. During the test case, three factors were observed: does reinforcement learning module work together with EV3 module of the physical computing framework, does the reinforcement agent learn actually and is it possible to use the `when` clause (introduced in Chapter 2.3) in JS-Eden with reinforcement learning module to provide automated learning process. The settings of the test case can be seen in Figure 23.

Figure 23. The settings of the test case of the reinforcement learning module and EV3
The state array only consisted in one state (the ultrasonic sensor value), so the level of complexity was not great. The ultrasonic sensor values of the EV3 module can range from 1 cm to 30 cm but sometimes the readings were negative. The negative values were ignored, because they were a result of the erroneous readings.

```python
motorA = "A";
motorB = "B";
motorAValue = 0;
motorBValue = 0;
ev3_motors_myMotors is [motorA, motorAValue, motorB, motorBValue];
ev3_us_sensor_2 = @;
value is ev3_us_sensor_2 - 200;
states is [value];
actions is ["fwd", "back", "left", "right"];  
hiddens = 4;
reinforcement_agent is [states, actions, hiddens];
epsilon = 0.2;
reinforcement_epsilon is epsilon;
reinforcement_gamma = 0.95;
reinforcement_iterations = 5;
reinforcement_experience = 10;
learn = false;
reinforcement_learn is learn;
reinforcement_action = @;
reward = 0;
reinforcement_reward is reward;
tick = 0;
```

Figure 24. Initialization construal of EV3 and reinforcement module
when (tick) {
    wait 200;
    learn = false; ## trigger prediction
    if (reinforcement_action == "fwd") {
        motorAValue = 30;
        motorBValue = 30;
    } else if (reinforcement_action == "back") {
        motorAValue = -30;
        motorBValue = -30;
    } else if (reinforcement_action == "left") {
        motorAValue = -30;
        motorBValue = 0;
    } else {
        motorBValue = -30;
        motorAValue = 0;
    }
    if (value > 6) { ## distance if less than 20 cms
        if (reinforcement_action == "fwd")
            reward = 0.9;
        else if (reinforcement_action != "back")
            reward = 0.8;
        else
            reward = -2;
    } else if (value <= 6 && value > 0) { ## skip weird, negative values
        if (reinforcement_action == "fwd")
            reward = -8;
        else if (reinforcement_action != "back")
            reward = -0.8;
        else
            reward = 0.8;
    }
    wait 200;
    learn = true; ## trigger q-value update
    tick++;
}

tick++;

Figure 25. Reinforcement construal for the test case
4.2.3 Test case results

During the test case, the parameters of the reinforcement learning module had to be changed (the number of iterations had to be dropped as well as the size of the experience) because the environment of the agent was so simple (only one input, four outputs). However because of the observable values in JS-Eden can be changed during the runtime, the change of the observable values effecting to the learning was easy. Also by opening the "Observable List", the learning process of the reinforcement agent was easy to debug; real time updates of the values of the current state, the current action of the reinforcement agent and the current reward were available and thus it was easy to make changes to number of the iterations, the experience and the learning rate.

The test case began with 30 as the number of iterations, 30 as the number of hidden units, 0.8 as the number of learning rate and 30 as the number of experience size. With these parameters the agent did not drive forward even if there were no obstacles ahead and the agent did not avoid obstacles but instead collapsed to the walls. The number of iterations, the experience size and the number of the hidden units were dropped and the learning rate was increased. The best result was achieved when the number of iterations was 5, the size of the experience was 10 and the number of the hidden units was 4. Quite small parameters worked the best because of the low complexity level of the environment; the reinforcement agent drove directly forward if there were no obstacles ahead and if the reinforcement agent encountered an obstacle, the reinforcement agent either turned left or right or drove backwards. The training took about 700 executions of the when clause.

Use of the when clause was possible and thus the EV3 robot explored its environment autonomously; the notations of the reinforcement construal (Figures 24 and 25) did not require any manual execution. The main challenges were that the readings from the ultrasonic sensor were sometimes negative and the values bounced from for instance 14 cms to 30 cms. The odd ultrasonic sensor values had an influence on learning because the reinforcement agent incorrectly learned that from (for instance) state 14 by using "forward" agent shifts to the state 30, which is of course not possible because the time between the executions of the when clause was
only less than a second and the EV3 robot moved about one cm. Also sometimes when the EV3 robot collapsed with the wall, the sensor value was 20 and the reinforcement agent continued to drive forward (as it should have been doing when the value was 20).

Furthermore the robotics client passed bound observable values to the EV3 server and the reinforcement learning module. The EV3 served on the other hand passed data from the EV3 brick to the client module and the reinforcement learning module passed data (the action from the action array) to the robotics client that updated the bound observable value.
5 DISCUSSION AND FUTURE WORK

In the discussion and future work chapter the thesis will discuss the challenges, the limitations and the advantages of the physical computing framework as well as the possible future applications of the physical computing framework. The possible future applications include both new construals and some already existing construals that could be extended to the physical world with Arduinos and EV3 robots.

5.1 Lessons learnt

The implementation, the evaluation and the design phases of the research process raised some challenges and discussion. The challenges mainly included the features of JS-Eden and the robotics platforms that had to be taken in the consideration during the research process. These challegnes will be discussed in this sub-chapter.

During the implementation and the evaluation phase of the each module of the physical computing framework, the nature of making construals had to be taken in the consideration: JS-Eden enables user to use \texttt{when} clauses to write automations that are triggered every time, the condition of \texttt{when} clause is true or a trigger observable has been changed. In order to provide a robotics framework for the users and enable the users still use all of the features of JS-Eden, the core functionality of JS-Eden had to be investigated and reverse engineered in order to enable the robotics client to ”sniff” the observable value changes from the JS-Eden core. The lack of a public API for extending the JS-Eden with a plug-in made the development of the physical computing framework quite challenging.

The development of the physical computing framework as well as the use of the physical computing framework requires some additional knowledge in electronics, especially the use and the development of the Arduino module. I lack the experience of the electronics and thus the development of Arduino module was the most challenging task during the research process.

The research process increased my technical knowledge in EM and JS-Eden as well as my technical knowledge in two robotics platforms (EV3 and Arduino). The
challenge was how to develop a robotics framework and fit it to the state-as-experienced philosophy of EM; for instance if a value of an observable that is responsible for controlling a motor of EV3 robot is dependent on a value of another observable and another observable value is changed, how to automatically rotate a motor.

5.2 Advantages and limitations of the physical computing framework

The purpose of the developed physical computing framework was to provide a complete robotics framework for the users of JS-Eden either to connect an EM artefact to the physical world with a robot or to create an EM artefact that would operate in a physical world as a robot. However if a user chooses to control either Arduino or EV3 robot with the physical computing framework, the user has to run a server locally for that purpose. The server has to be run locally because JS-Eden is a browser based environment written in JavaScript and JavaScript in the browsers runs in the isolated sandbox with no access to the file system (serial ports). Setting up an EV3 server or an Arduino server can be time consuming and requires NodeJS to be installed on a host computer. Also if the user uses a USB-communication channel between the Arduino board and the Arduino server, the user has to know which serial port is used by the Arduino to specify the serial port address to the Arduino Server. Furthermore the Arduino server’s communication with the Arduino board requires the ino-file from the physical computing framework directory to be compiled and uploaded to the Arduino board.

However the physical computing framework is not isolated in any way from JS-Eden so the user can fully operate in JS-Eden while controlling a robot for instance. Also the user can modify the EM artefact on-the-fly without the need of uploading a newly written script to a robot (which has to be done if the user were to use Arduinos or EV3s in a traditional way). Scratch (a visual programming language for educational purposes) can also be used to control Arduino and EV3 so that the construals can be modified on-the-fly (in this context "on-the-fly" means that the construal is not always re-compiled and re-uploaded to the robot) but it should be
noted that Scratch and JS-Eden tend to be designed for the different age-groups. The identified limitations of the physical computing framework are:

- **Setting up an Arduino or EV3 server requires external components and some form of experience of them**: NodeJS has to be installed to a host computer as well, as the JS-Eden has to be run locally (the physical computing framework cannot be used in the instance of JS-Eden that is hosted by Warwick). Also a special ino-file (which ships with the physical computing framework) has to be compiled and uploaded to Arduino with the Arduino IDE.

- Sensor values from the Arduino board and EV3 are sometimes incorrect (for instance ultrasonic sensor values from EV3 robot have been incorrect)

On the other hand, the physical computing framework tends to have some advantages over the existing open-source plug-ins, that are used to control either EV3 or Arduino with a programming language:

- Both Arduino and EV3 support included as well as the feature to simulate the robot movements in 3D space without an actual physical robot

- JS-Eden does not require write-and-execute type of approach but the controls of the robot can take place on-the-fly

- The physical computing framework is not separated from the rest of the JS-Eden: The full libraries of JS-Eden as well as the models are available for the robotics notations.

### 5.3 Future work

Soon after the SciFest 2016, it was discussed in the Construit project group that developing Arduinolas should be continued. It was decided that the Arduinola construal should be divided into the multiple agents, where other agents could handle the looks of Arduinola in the simulator window whereas other agents would handle the level of happiness of Arduinola. In this way not too much of the Eden code would be exposed to the children at the same time and it probable would be easier to make the changes to the Arduinola construal.
During the SciFest 2016 only two different sensors were available for each Arduinola. Testing other analog sensors could help to make making Arduinola happier by moving it around more challenging. It was also discussed that Arduinolas could be merged with the existing construals found in the repository. Such construals include the theremin (from the repository of JS-Eden's version control system), which could be used to include voices in Arduinolas that could indicate the level of happiness.

One suggestion was that instead of one Arduinola in the simulator window, Arduinolas could be merged with the nim construal. Nim is a mathematical strategy game where two players remove objects from the heap during the player's own turn. Merging the idea of nim to the group of Arduinolas in the simulator window could create more challenging settings for the children toying with the Arduinola construal and the physical Arduinola. Together with the idea of the nim construal and the theremin construal, the Arduinola construal could be extended to include sounds and the abstract mathematical thinking that would eventually connect three different construals and link the merged construal to the physical world. By adding other sensors besides the sensors that were used during the SciFest 2016 (the temperature and humidity sensor and lighting sensor) to the merged Arduinola construal could also increase the breadth of Arduinolas.

Besides the concept of Arduinola, the physical computing framework could be used to create a wall following robot (either Arduino or EV3) that would be controlled remotely by the physical computing framework. The wall following is somehow challenging task for the programmer since the sensor values and the motor powers need to be optimized well in order to success. If the task is accomplished by traditional ”write a program, compile, upload, test, change” -approach, that task can be time consuming. Yet the ”change on-the-fly” philosophy of the physical computing framework could help to develop the wall following robot in much less time. The wall following construal could also include the usage of the reinforcement learning module; for instance the agent would be rewarded if it controlled the motors of the robot (either Arduino or EV3) to stay a certain distance away from the walls.

Currently, the Arduino module has been tested with two different Bluetooth modules and both of them were usable. Although the baud rates varied: the other one worked
with 9800 baud rate and the other worked with 36000 baud rate. The change to baud rates has to be done currently to ino-file of Arduino so it is recommended to use USB cable if the user does not have any experience in C programming language.

It was discussed that the physical computing framework version of JS-Eden would be its own branch that would be available in GitHub. During the SciFest 2016 the tutors did not notice any bugs in the physical computing framework but the constant usage of the Bluetooth module drained the battery quite often so that the Bluetooth connection between the PC and the Arduino board was not always very stable.

The reinforcement learning module should be tested with a more complex environment. The test case described in chapter 4.2.2 could be extended by adding additional sensors to the EV3 robot to build a construal for a robot following the wall and navigating in a maze for instance. For that case, there should be at least two ultrasonic sensors: one facing either left or right and one facing forward. If one chooses to increase a level of complexity of the environment (to increase the number of inputs and outputs), one also should increase the number of the hidden units, the number of iterations and the size of the experience.

Currently the physical computing framework enables only one device (EV3 robot or Arduino board) to be connected to JS-Eden. An interesting approach in the future would be enabling the framework to communicate with multiple devices in Internet of Things (IoT) manner to gather sensor data. IoT approach could be combined with machine learning in order to develop construals to explore and understand analyzed sensor data that is gathered from multiple IoT devices connected to JS-Eden.
6 Conclusions

During the Construit!, JS-Eden has gained quite a few extensions and the great deal of effort has been put into the development the user experience of the JS-Eden as well as into the Eden syntax. Important developments can be seen for instance in the update of the ternary statements into the more readable form. Another update is to allow users to execute the Eden notations in any way the users want to. The Construit! will end in the fall of 2017.

The physical computing framework offers a flexible way to develop physical computing construals in Eden to control either Arduinos or EV3 robots as well as to develop 3D simulations of the robots and integrate a machine learning agent in the robotic construal. Yet it is clear that the installation process and the use of either EV3 or Arduino is challenging for the people who have no previous knowledge in NodeJS. Also the use of the simulator module or the reinforcement learning module requires some previous knowledge in the Eden syntax. Thus it is recommended that before a new user starts using the physical computing framework, there should be a short introduction to JS-Eden.

All of the modules of the physical computing framework were tested and evaluated during the research process. The physical computing framework modules can be used as a stand-alone modules but they can also be used together (for instance the simulator module can be used together with Arduino module).

In the introduction chapter, the thesis provided two research questions and during the following chapters, the thesis attempted to provide answers to those questions. The following will provide sum up about the research questions and the concrete answers for the research questions:

**RQ1: Is it technically feasible to implement a complete robotics framework for the existing EM platform?**

It is indeed technically feasible to implement a complete robotics framework for JS-Eden and while using the framework (the physical computing framework) it is still
possible to present the construals in the EM manner. Although it should be noted that the implementation of the physical computing framework required some reverse engineering of JS-Eden in order to provide the automatic observable listening function for the physical computing framework in case of for instance the user wants to use when-clauses instead of a manual execution of the statements.

**RQ2: What has to be considered in the implementation?**

The syntax of Eden currently allows user to write construals so that the construal is either manually executed line by line or that the whole construal is run as a whole. In order to retain both ways of making construals while using the physical computing framework, the implementation of the physical computing framework had to be inclusive for both options. Also the automatic updates of the observables bound to the for instance Arduino board through dependencies had to be included in the physical computing framework.

**RQ3: What is the benefit of bringing robotics as part of making construals activity?**

The physical computing framework provides a wide range of possible use-cases for those who have an interest in robotics. Also the physical computing framework provides a possibility to increase the range of possible construals by allowing user to extend a construal to the physical world or from 2D canvas picture to 3D simulation.
REFERENCES


[7] Empirical Modelling, Basics of Eden; retrieved from:

https://www2.warwick.ac.uk/fac/sci/dcs/research/em/teaching/cs405/session2.3.pdf (May 23 2016)

[8] Empirical Modelling, Construit! Meeting time-table; retrieved from:

https://www2.warwick.ac.uk/fac/sci/dcs/research/em/construit/year2/c6/ (May 23 2016)

[9] Empirical Modelling, Eden language; retrieved from:

https://www2.warwick.ac.uk/fac/sci/dcs/research/em/software/eden/ (May 23 2016)

[10] Empirical Modelling, JS-Eden; retrieved from:
https://www2.warwick.ac.uk/fac/sci/dcs/research/em/software/js-eden/ (May 23 2016)

[11] Empirical Modelling principles; retrieved from:
https://www2.warwick.ac.uk/fac/sci/dcs/research/em/intro/principles/ (May 23 2016)

[12] Empirical modelling software; retrieved from:
https://www2.warwick.ac.uk/fac/sci/dcs/research/em/software/ (May 23 2016)

[13] Empirical Modelling, state-as-experiences; retrieved from:
https://www2.warwick.ac.uk/fac/sci/dcs/research/em/intro/glossary/#stateasexp (May 23 2016)

[14] Empirical Modelling tools, WebEden; retrieved from:
https://www2.warwick.ac.uk/fac/sci/dcs/research/em/software/webeden/ (May 23 2016)

[15] EV3 serial port communication protocol; retrieved from:
https://github.com/hflw/node-robot/tree/master/lib/ev3
(June 21 2016)


[21] JS-Eden plugin: observable mining; retrieved from:


[22] JS-Eden plugin: speech synthesis; retrieved from:


[23] JS-Eden source code; retrieved from:

https://github.com/EMGroup/js-eden (May 23 2016)


[26] ThreeJS JavaScript based WebGL engine web page; retrieved from

http://threejs.org/ (May 25 2016)