

# Simulation and Rapid Prototyping for Mechatronic Design

DIPLOMARBEIT

zur Erlangung des akademischen Grades

**Diplom-Ingenieur**

im Rahmen des Studiums

**Wirtschaftsingenieurwesen Informatik**

eingereicht von

**Gerhard Brunner**

Matrikelnummer 0430210

an der  
Fakultät für Informatik der Technischen Universität Wien

Betreuung: Ass. Prof. Dipl.-Ing. Dr. techn. Monika Di Angelo  
Mitwirkung: Ass. Prof. Dipl.-Ing. Dr. techn. Peter Ferschin

Wien, 09.11.2013

\_\_\_\_\_  
(Unterschrift Gerhard Brunner)

\_\_\_\_\_  
(Unterschrift Betreuung)

# Simulation and Rapid Prototyping for Mechatronic Design

MASTER'S THESIS

submitted in partial fulfillment of the requirements for the degree of

**Diplom-Ingenieur**

in

**Business Engineering and Computer Science**

by

**Gerhard Brunner**

Registration Number 0430210

to the Faculty of Informatics  
at the Vienna University of Technology

Advisor: Ass. Prof. Dipl.-Ing. Dr. techn. Monika Di Angelo

Assistance: Ass. Prof. Dipl.-Ing. Dr. techn. Peter Ferschin

Vienna, 09.11.2013

\_\_\_\_\_  
(Signature of Author)

\_\_\_\_\_  
(Signature of Advisor)

# Erklärung zur Verfassung der Arbeit

Gerhard Brunner  
Muggauberg 23, 8152 Stallhofen

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

---

(Ort, Datum)

---

(Unterschrift Gerhard Brunner)

# Acknowledgements

First I want to thank my parents for their trust and support during my time of being a student in Graz and Vienna. My interest in technical and business topics goes back to my father Wolfgang, who has always influenced me in a very positive way during talks and discussions. I also want thank my mother Margit for welcoming me at home and being supportive.

I gratefully thank Prof. Monika Di Angelo and Prof. Peter Ferschin for supporting and helping me on this master thesis. They inspired me with their idea of having an easy to use rapid prototyping environment for mechatronic simulations.

At last I also want to thank my friends who have joined me during the time of school and study.

# Abstract

Mechatronics is an interdisciplinary field which combines software, electronics, mechanic and control theory. It is applied in different areas like industrial assembly, robotics and consumer products.

Each of the interdisciplinary fields has a high complexity which requires engineers with knowledge of specific area of expertise. Additionally the development of interfaces between the different fields requires a high communication effort of the engineers which results in even higher costs of time and money. In the product development cycle prototypes are used for solving technical problems and usability issues in an early development phase. Technical problems and usability issues which are not discovered and solved in this prototyping stage are causing higher costs in the later stages.

This work evaluates development environments which can be used for designing mechatronic applications in a rapid prototyping approach. Thereby the focus is set on virtual development environments which simulate kinematic behavior. The environments differ in their focus on certain aspects of mechatronics and in their benefit of doing rapid prototyping.

In summary, the development environments are either too complex for rapid prototyping of mechatronics, or are lacking important features.

Therefore a rapid prototyping development environment for simulating and controlling mechatronics is designed. In comparison to the presented environments, it allows to rapidly design mechatronic applications without the need of programming and electronic skills. It implements a simulation environment programmed with the Unity game engine for simulating fischertechnik components. Additionally, the mechatronic behavior of the virtual prototype can be transmitted to the physical prototype via an arduino board.

**Keywords:** mechatronics; virtual simulation in 3D; rapid prototyping; kinematic design; fischertechnik, arduino, Unity game engine.

# Kurzfassung

Mechatronik ist ein interdisziplinäres Forschungs- und Anwendungsgebiet aus den Feldern Software, Elektronik, Mechanik und Regelungstechnik. Sie findet heutzutage Anwendung in unterschiedlichen Bereichen wie der industriellen Fertigung, Robotik und Konsumentenprodukte.

Jedes der interdisziplinären Felder weist eine hohe Komplexität auf und erfordert Spezialisten aus den entsprechenden Gebieten. Durch die Notwendigkeit der Schnittstellenentwicklung zwischen den Feldern entsteht zwischen den Fachspezialisten ein erhöhter Kommunikationsaufwand. Diese Faktoren führen zu hohen Entwicklungs- und Zeitkosten bei der Entwicklung mechatronischer Systeme. Im Produktentwicklungszyklus ist die Prototypenentwicklung die erste technische Entwicklungsphase in denen potentielle Probleme gelöst, und Benutzbarkeit validiert werden. Technische Probleme welche in dieser Phase nicht erkannt werden verursachen höhere Kosten in den nachgelagerten Entwicklungsphasen.

In dieser Arbeit werden Entwicklungsumgebungen für mechatronische Prototypen evaluiert. Der Fokus liegt dabei auf virtuellen Entwicklungsumgebungen welche kinematisches Verhalten simulieren. Die Entwicklungsumgebungen unterscheiden sich in ihrem Fokus auf bestimmte Aspekte der Mechatronik, Komplexität und in ihrer Möglichkeit Rapid Prototyping zu betreiben. Eine Evaluierung dieser Umgebungen führt zum Ergebnis dass die virtuellen Simulationsumgebungen entweder eine zu hohe Anwendungskomplexität besitzen, oder zu wenig Features aufweisen.

Daher wird eine Entwicklungsumgebung entworfen, welche den Aufbau und eine virtuelle Simulation von mechatronischen Anwendungen ermöglicht. Verglichen mit den evaluierten Entwicklungsumgebungen erlaubt es diese, mechatronische Anwendungen mittels Rapid Prototyping zu entwickeln ohne dass dabei Programmier oder Elektronikkenntnisse benötigt werden. Die Implementation erfolgt mit der Unity game engine, Arduino und Fischertechnik. Der virtuelle Prototyp kann zusätzlich an den realen Prototypen gekoppelt werden. Somit lässt sich das kinematische Verhalten aus der virtuellen Simulation heraus direkt auf den physikalischen Prototypen übertragen.

**Schlagerworte:** Mechatronik; Virtuelle Simulation in 3D; Rapid Prototyping; Kinematisches Design; Arduino; Unity game engine.

# Contents

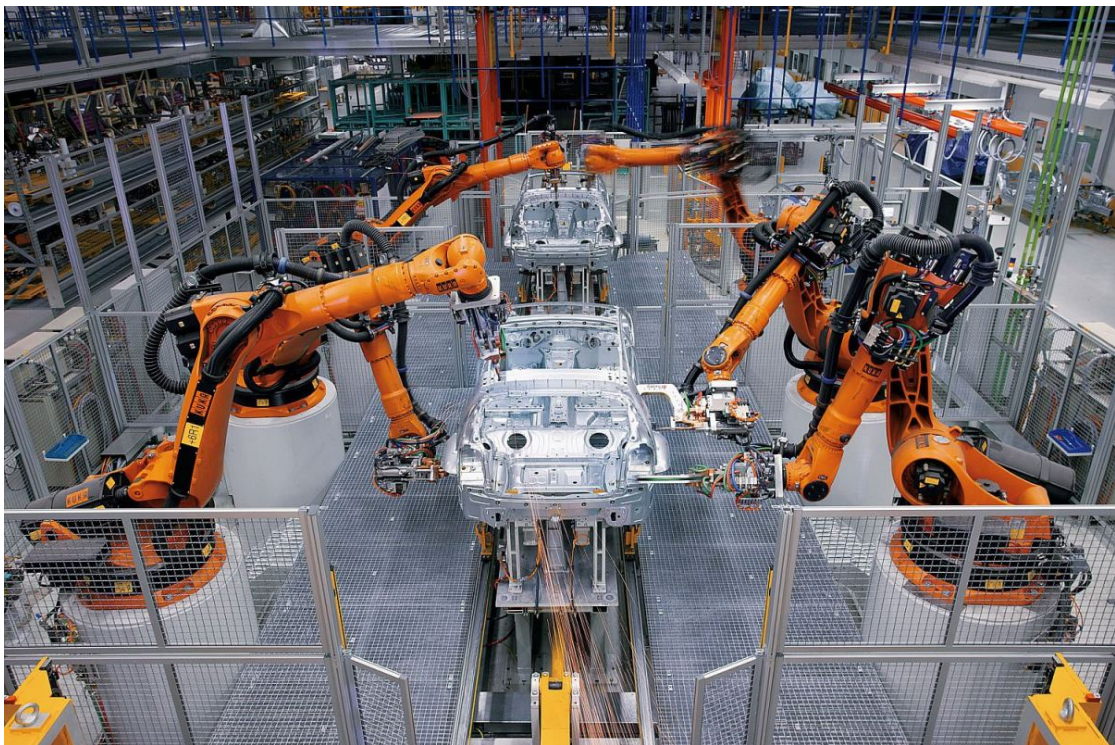
<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Motivation . . . . .	1
	Simulation for Mechatronic Design . . . . .	2
	Rapid Prototyping . . . . .	4
	Costs in the Development Process . . . . .	5
1.2	Research Question . . . . .	6
1.3	Aim of the Work . . . . .	6
1.4	Methodological Approach and Structure of the Work . . . . .	7
<b>2</b>	<b>Survey of Prototyping Approaches</b>	<b>8</b>
2.1	State of the Art . . . . .	8
	Focus on the Prototyping Process . . . . .	8
	Focus on Simulation and Control . . . . .	9
	Focus on Interfaces . . . . .	9
	Focus on Education . . . . .	9
2.2	Evaluation Criteria . . . . .	10
	Prototyping Process . . . . .	10
	Prototyping of Mechatronic Systems . . . . .	11
	Hardware Components . . . . .	11
	Software Environment . . . . .	11
	Interfaces of the System Architecture . . . . .	11
	Visual Authoring . . . . .	11
	Transmission of mechatronic behavior . . . . .	12
	Usability . . . . .	12
	Economic Criteria . . . . .	12
2.3	Prototyping Environments . . . . .	12
	Fischertechnik . . . . .	12
	Fischertechnik Designer . . . . .	14
	Creo Parametric Software simulating Motion . . . . .	16
	Arduino . . . . .	16
	Phidgets . . . . .	18
	d.tools . . . . .	20
	Phybots . . . . .	23

LEGO Mindstorms NXT . . . . .	25
Microsoft Robotics Developer Studio . . . . .	30
2.4 Comparison and Summary of Existing Approaches . . . . .	34
Overview . . . . .	35
Detailed Evaluation . . . . .	35
Research Conclusion . . . . .	39
<b>3 Concept for a Mechatronic Design Environment</b>	<b>41</b>
3.1 Concept . . . . .	42
3D design environment . . . . .	43
Communication interface . . . . .	44
Physical design environment . . . . .	45
3.2 Design and Implementation . . . . .	45
Main Components of the proposed Environment . . . . .	45
Principles of the Unity Game Engine . . . . .	47
Specification of the Simulation Environment . . . . .	48
Implementation of the Simulation Environment . . . . .	52
Specification of Fischertechnik Synchronization . . . . .	64
Implementation of Fischertechnik Synchronization . . . . .	64
<b>4 Evaluation and Critical Reflection</b>	<b>68</b>
4.1 Evaluation on the Example of a 2D Plotter . . . . .	68
Overview of the Prototyping Process . . . . .	68
Preparation . . . . .	69
First prototypical Iteration . . . . .	71
Second prototypical Iteration . . . . .	76
4.2 Comparison with Related Work . . . . .	80
<b>5 Summary and future work</b>	<b>82</b>
<b>Bibliography</b>	<b>84</b>



# Introduction

## 1.1 Motivation



**Figure 1.1:** Spot welding on an assembly line with KUKA robots

Mechatronic applications are widely used in the manufacturing industries. Since the industrial revolution the degree of human work in the manufacturing process decreases while the degree of machine work increases. The product life cycles are getting shorter and the market asks for increasingly faster innovation cycles. In mass production these products are often assembled by mechatronic robots which are part of a mechatronic assembly line. Further, the product itself might also be a mechatronic application (figure 1.1).

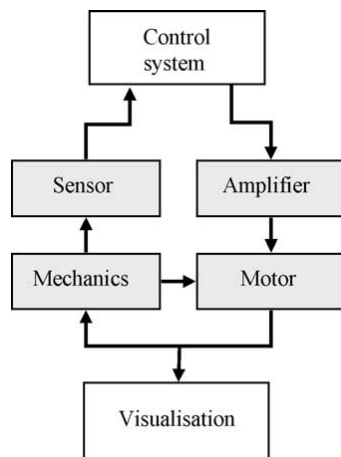
Advancing technology, new products ideas and changing customer demands results in new products and therefore in continuous changes of the assembly process. To shorten the entire product development cycle, solutions to increasing the speed of all product development and product manufacturing stages are requested. When developing new products or manufacturing processes, specific problems and challenges have to be solved early. The earlier problems are discovered and solved the lower are the costs to handle them in the later developing stages. Therefore prototypes are used. Since manufacturing lines are mechatronic system and the products can be mechatronic applications as well, appropriate prototypes are increasing the speed of the development- and manufacturing process.

“Mechatronics is an interdisciplinary field of engineering that integrates design techniques in precision mechanical engineering, control theory, computer science, and electronics into the overall design process“. [58]

Each of these fields itself is complex and requires highly skilled professionals. Developing new mechatronic applications is cost and time intensive. When these fields are combined for developing mechatronic applications, the complexity increases significantly. The impact of errors is increased because one dysfunctional field blocks all the others. Besides the accumulating complexity of technology, due to the requirement of technical interfaces between the fields, the communication expenditure between people also increases.

### **Simulation for Mechatronic Design**

“The mechatronic system can exist in reality or as the complex system of a simulation model“. [7]. Banks defines *Simulation* as: “... the imitation of the operation of a real-world process or system over time. Simulation involves the generation of an artificial history of the system and the observation of that artificial system that is represented.“ [4]



**Figure 1.2:** Simulation model for mechatronics which visualizes simulated sensors, amplifiers, mechanics and motors (actuators). [32]

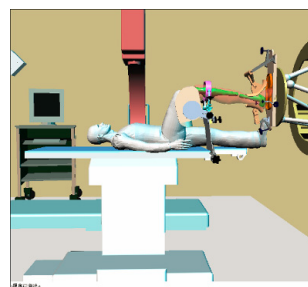
In figure 1.2 the *sensor*, *amplifier*, *mechanics* and *motor* are representing this imitation of the real world. Ideally the *control system* can be used for simulated, as well as for physical sensors and motors. In a simulation a *visualization* is required which can range from simple text logs to real time graphic representations in 3D.

### Simulated Mechatronic Systems

Medics, pilots or employees of automated factories are examples of people who are using mechatronic applications in their job. Errors executed on these mechatronic applications can cause major risk for human life. Therefore virtual simulations for training and education of such mechatronic systems are developed. Figure 1.3 depicts the virtual cockpit view of an air plane. The purpose of the simulation in figure 1.4 is to train for advanced navigation in surgeries. Figure 1.5 depicts a controllable underwater glider in a virtual simulation.



**Figure 1.3:** flight simulator [52]



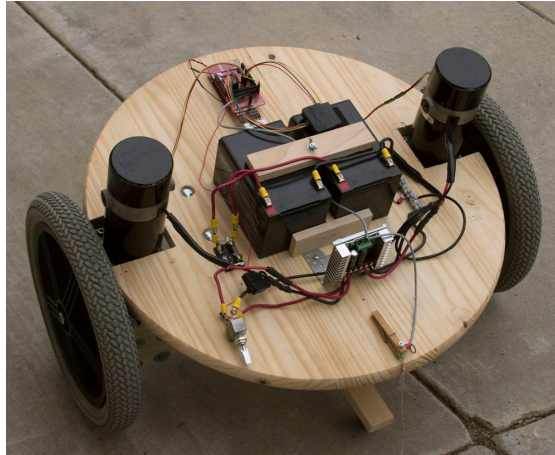
**Figure 1.4:** medical surgery [2]



**Figure 1.5:** underwater glider [59]

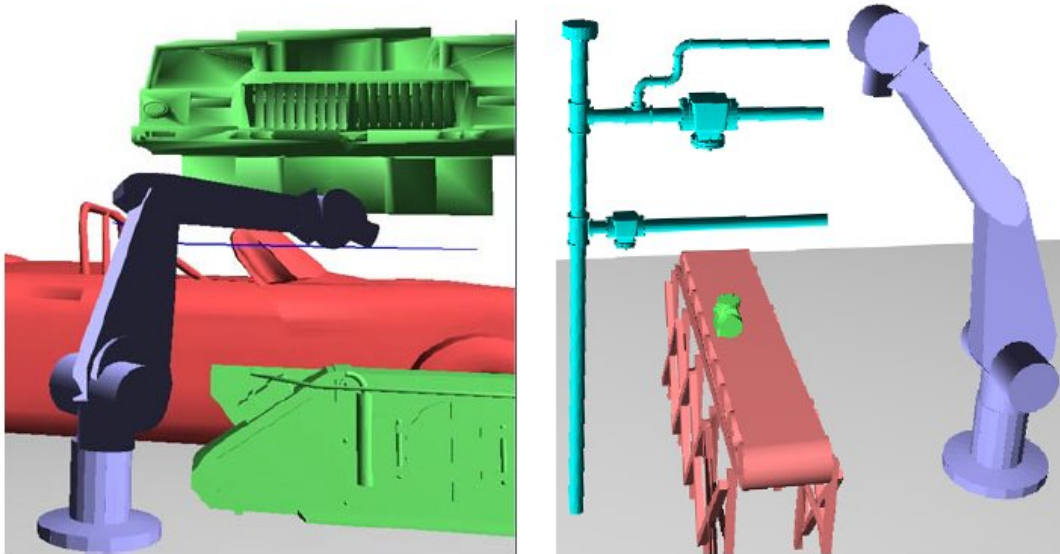
## Rapid Prototyping

A *Prototype*, as a result of prototyping, is defined as “Prototype: an early or original form; ... (in engineering) a full-scale model of a structure or piece of equipment, used in evaluating form, design, fit, and performance.” [46]. The Oxford Advanced Learner’s Dictionary defines a *Prototype* as “the first design of something from which other forms are copied or developed”. [29]



**Figure 1.6:** Example prototype of a Segway [28]

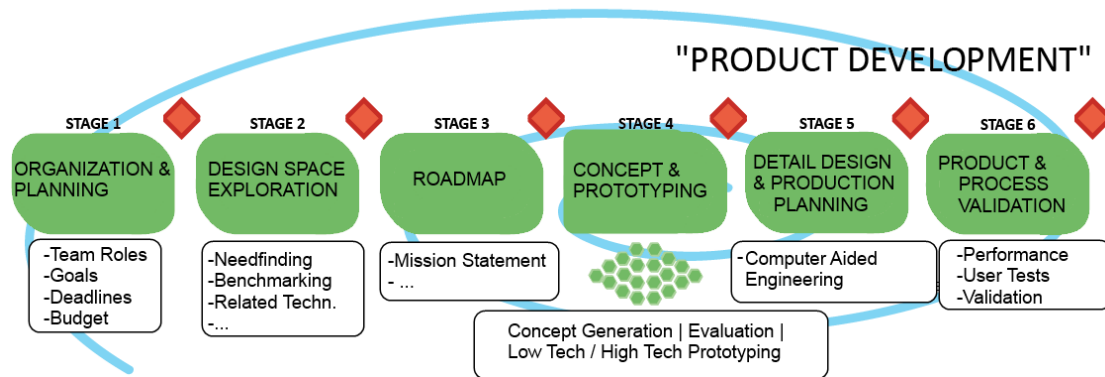
In contrast *Virtual Prototyping* does not produce a physical object. Wang defines a Virtual Prototype as: “... a computer simulation of a physical product that can be presented, analyzed, and tested from concerned product life-cycle aspects such as design/engineering, manufacturing, service, and recycling as on a real physical model. The construction and testing of a virtual prototype is called virtual prototyping (VP).” [55] An example of virtual prototyping for industries is figured in 1.7.



**Figure 1.7:** In this example the required movement of industrial robot arms is discovered by rapid prototyping in a virtual environment. On the left: Automated Car Painting Scene where the robot arm follows a path over the car body while avoiding obstacles. On the right: Assembly Line Planning Scene where the robot arm avoids the moving pipes to reach a moving part passing on the conveyer belt. [19]

### Costs in the Development Process

Figure 1.8 depicts the stages of product development. The blue line indicates that changing requirements or occurring obstacles in later stages requires an additional iteration of the previous stages. Therefore, the later product changes are demanded, the higher are the production costs and time of development. The first three steps (organization & planning, design space exploration, roadmap) are defining the organizational aspects, the development environment and the product specifications. At stage four the technical development starts by creating prototypes for solving technical problems and for doing first user tests. In the fifth stage the outcomes of the prototyping stage are taken for generating a detailed and full production plan. The last stage defines a validation of the product and the process. The product development is the most cost intensive stage. Therefore problematic issues which have not been solved in the prototyping stage are generating enormous costs in the production stage.



**Figure 1.8:** The product development cycle. [24]

## 1.2 Research Question

Concluding, costs can be saved by recognizing and solving problems in the beginning of the product development cycle. Prototyping is the first step where features are brought into a functional version of the product. Further, prototyping is also used in development and setup of manufacturing applications. Besides physical prototyping, virtual prototyping is a step further into saving costs and is also used for training and education. This leads to follow research question:

*In which ways can rapid prototyping and simulation be applied for mechatronic design?*

## 1.3 Aim of the Work

On the one hand developing mechatronic applications is costs intensive due to its interdisciplinarity. And on the other hand solving problems in the prototyping stage saves a lot of costs for the detail planning and the production stage of a product. The costs for prototyping can be decreased further by having an environment for developing virtual prototypes. “Virtual prototyping is described as a prototyping process, in which a product or a product concept, its behavior and usage situation is simulated as realistically as possible by using computer models and virtual reality techniques“. [38]

Therefore the goal for this work is to come up with different ways of prototyping mechatronic applications. The focus lies on environments which support the development of virtual prototypes. To meet also the requirements in the sector of education and training, the usability and complexity of these environments have to be suitable for users which are not experts in the field of mechatronics.

## 1.4 Methodological Approach and Structure of the Work

The *Introduction* 1 gives a motivation for the requirement of having a design environment for mechatronic prototypes. The required features for a development environment for mechatronic applications are defined in the section *Evaluation Criteria* 2.2.

Literature and work in the field of mechatronics, rapid prototyping, simulation, robotics or physical interface that covers the required features is collected and summarized in the chapter *Survey of Prototyping Approaches* 2.

The summarized literature and work is evaluated based on the evaluation criteria and the outcome of this evaluation leads to suggestions for environments that feature rapid prototyping for mechatronic applications (section *Comparison and Summary of Existing Approaches* 2.4).

A new development environment for developing mechatronic prototypes is presented and described in chapter *Concept for a Mechatronic Design Environment* 3). This concept for a new mechatronic design environment id then be implemented (section *Design and Implementation* 3.1). To evaluate the implemented concept a 2D plotter is designed as a virtual and physical prototype and evaluated in chapter *Evaluation and Critical Reflection* 4.

# Survey of Prototyping Approaches

This chapter gives an overview and evaluation of the current state-of-the-art in the field of rapid prototyping for mechatronic design. The section *State of the Art* (2.1) describes work of science and industry which covers mechatronics in combination with rapid prototyping or virtual simulation. From the section *State of the Art* the *Evaluation Critearia* (2.2) for rapid prototyping environments for mechatronic applications are concluded. These evaluation criteria cover the integral components of rapid prototyping systems for mechatronic applications. In section 2.3 several applications that already feature rapid prototyping are discussed. Section 2.4 presents a detailed evaluation of available applications where the criteria were applied onto the prototyping environments.

## 2.1 State of the Art

### Focus on the Prototyping Process

“Reflective practice, the framing and evaluation of a design challenge by working it through, rather than just thinking it through ...“ is the key aspect in prototyping, concluded by Hartmann [26]. Hartmann describes iterations as a central concern in a design process which comes up with prototypes in every iteration. The environment d.tools enables early stage prototyping and an evaluation of the prototyping process of physical interfaces. [26]

Digital mock-ups (DMU), i.e. “...a realistic computer simulation of a product with the capability of all required functionalities...” [12], are used in the car manufacturing in combination with virtual reality and virtual prototyping [12] [18].

In [22] a cooperate assembly environment for designing virtual mechatronic prototypes is presented. It includes a 3D virtual reality environment for evaluation and a 2D touchscreen workbench where the mechatronic control (with Matlab/Simulink) is defined.



## **Focus on Simulation and Control**

In the manufacturing industries 3D software environments are used for prototyping and simulation (e.g. computer aided robotic (CAR) [11]) for industrial robot cells). Such software environments are used to prepare workflows for robot workcells on the software side. A general solution for automatic code generation for physical robot workcells resulting from the simulations is currently not available. Engineers have to implement the workflow of the simulated control again on the physical side. [44]. Moore presents an integration of the simulation environment and control system environment where simulated control can be transferred seamlessly to the control system. The control logic is defined by a programming environment [31] within the simulation environment. [44]

Jönsson et.al describes, implements and validates the concept of a real time simulation of computerized numerical control (CNC) machine tools. It encompasses a control system, simulation models of the kinematics and a 3D visualization. For validation a simulation of a water jet cutting machine was chosen. The 3D simulation is based on the computer aided design (CAD) document of the manufacturer. A validation of the simulated cutting machine compared to the physical cutting machine indicates deviations up to 0.5 mm. [32]

## **Focus on Interfaces**

Greenberg [23] states that developing simple interfaces between physical devices and the programming language on the PC is still an obstacle. Using simple hardware devices (e.g. sensors, switches, motors) requires basic knowledge in electronics. In comparison devices for the commercial mass market are working with plug & play but offer no application-programming interface (API). Hacking and reverse engineering are methods to achieve programming access to these hardware devices (e.g. Microsoft's Actimates hacked [35]; Lego Mindstorms RCX documentation [40]). Greenberg also states that on the other hand some devices (like programmable logic controllers) offer an API but the required programming skill can be very specific. Developers may also not have the required devices on hand. Due to shipping delays, costs or limited units developers may be able to design their applications only within a software environment. [23]

## **Focus on Education**

A low-cost and low level mechatronics prototyping environment for education is presented in [25]. It aims laboratory courses where programming and electronics are combined for building mechatronic prototypes.

Another prototyping environment for student courses which is build upon existing modules (Matlab/Simulink, Arduino, iRobotCreate) is described in [20]. Xie et. al. describes a virtual 3D design environment for Fischertechnik in [61]. A simulation software (Pro/E software) is used for defining the motion behavior. Its purpose is to learn students the process of mechanical design.

## 2.2 Evaluation Criteria

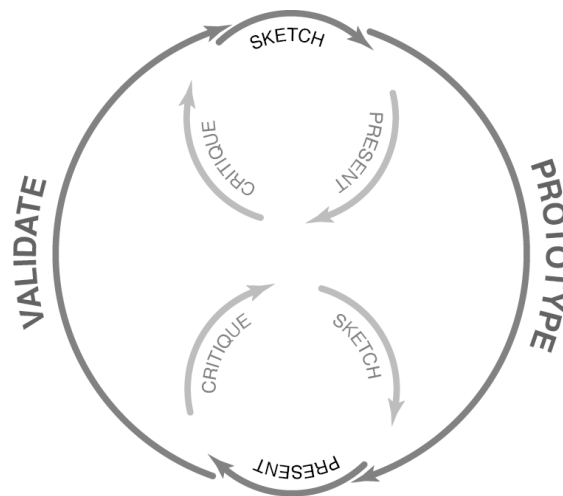
In section State of the Art (2.1) the development of prototypes is achieved with frameworks or environments which encompasses specific methods and components. These methods and components can be reformulated to evaluation criteria for prototyping environments. The main criteria are: *prototyping process*, *prototyping of mechatronic systems*, *hardware components* and *software environment*.

The prototyping processes may be enhanced by various framework features as: the option to simulate the kinematic behavior in a virtual simulation environment (*virtual simulation*); the option to transmit the simulated kinematic behavior to the physical prototype (*transmission of kinematic behavior*). Further aspects are *interfaces of the system architecture*, and *usability*, *price*, *availability*, *popularity* and *user support* of a prototyping environment.

### Prototyping Process

This core criterion checks if the evaluated prototyping environments can fulfill the prototyping process defined by Warfel [56].

Figure 2.1 depicts the iterative and evolutionary prototyping process. Sketching is the process to bring the ideas on media like paper, software simulation or whiteboards. A sketch is then evaluated with critique and possible approve from one or more persons. Prototyping is the part where an approved sketch is built as a physical model. This model is then being validated.



**Figure 2.1:** Prototyping process [56]

An exemplified process run is: “We sketch, present and critique, prototype, present and critique, sketch, present and critique, prototype, present and critique, prototype, and test. Then we do it all again.” [56]

## **Prototyping of Mechatronic Systems**

The prototypes in this thesis result from rapid prototyping environments, which are not static. So this criterion states that the prototyping environment will be evaluated on the possibility to build mechatronic prototypes. The environments must therefore feature an intuitive way to arbitrarily combine pieces that are static and pieces that can execute or support motion (e.g. motor, hinge).

## **Hardware Components**

The criteria for hardware components evaluate the features and complexity of the electronic environment. Actuators and sensors are connected to an electronic control unit (e.g. circuit board with processor, A/D converter, pins). This control unit executes reading and writing operations on actuators and sensors. For this criterion the following factors come into play: complexity and variation of electronic interfaces (e.g. I2C, A/D converter, wireless interface), role and complexity of processors (amount of processors, programming language, is the processor a data exchanger or does it fully control the prototype), extensions to hardware (additional electronics like amplifier, capacitor or transistor) and portability (e.g. the complexity of a new hardware setup with PC, circuit board with processor, power supply).

## **Software Environment**

Actuators and sensors are controlled by software either running on a PC, a microcontroller or on a PC together with a microcontroller. The software environment as a criteria include the factors: complexity as a feature (i.e. in which amount, variation and detail actuators and sensors can be controlled), complexity in usability (i.e. programmable control vs. GUI control) and portability (the complexity of software setup on a new PC and a microcontroller)

## **Interfaces of the System Architecture**

The software engineering institute (SEI) defines system architecture as: "... describing the elements and interactions of a complete system including its hardware elements and its software elements". [53] Interactions within hardware and software are done via interfaces. Therefore the interfaces of the system architecture are evaluated in terms of extensibility (is it possible to extend parts of the system with less effort), simplicity (the effort of attaching hardware or software parts to the provided interfaces) and robustness (e.g. can hardware be destroyed by wrong connections; does the software behave comprehensible and expected on use of interfaces).

## **Visual Authoring**

One aim of the Rapid Prototyping process is to simulate the prototype on the computer. This requires a software which features construction of 3D prototypes with kinematic behavior, physics simulation and the possibility to control the kinematic behavior. Furthermore it is evaluated if the software on the PC features 3D modeling and 3D control of the prototype. 2D control (e.g. statecharts) and a feedback system which responses events to the user are also evaluated.

## **Transmission of mechatronic behavior**

Mechatronic behavior for sensors and actuators can be programmed, designed in a visual tool (2D/3D) or controlled in real-time. The behavior is then stored in a digital format and has to be transmitted to the physical prototype. This criteria evaluates the complexity and effort for users to transmit simulation behavior onto prototyping environments. An example for high complexity and high effort is a prototyping system where written code has to be compiled first and then loaded on a microcontroller. A low complexity system executes mechatronic behavior in real-time on the physical prototype.

## **Usability**

Setting up a virtual prototype with a 3D software requires two features: first, a way to build the virtual prototype and second, a way to give kinematic instructions and behavior for sensors to the virtual prototype. The evaluation of this criteria involves understandability (does the user understand the use of the system), operating options (how intuitive is the realization of the prototyping process ) and attractiveness (aesthetic appearance of the software's components like buttons or layout; look and feel of the hardware)

## **Economic Criteria**

The economic criteria comprises price (of software and hardware), availability (e.g. delivery times of hardware, download sources of software), popularity (how well known the software or hardware is) and support (does software or hardware provide future bug-fixes or versions).

## **2.3 Prototyping Environments**

The environments, published papers, products and tools presented in this section support, describe, or fully integrate rapid prototyping for mechatronic simulations. For each prototyping environment, evaluation criteria factors described in section 2.2 are taken into account and assessed in the section comparison<sup>2.4</sup>.

### **Fischertechnik**

Published by the company Fischertechnik GmbH<sup>1</sup>, the product fischertechnik is a construction set with programmable mechatronics. The span of application includes playing for kids, education in school, courses in university, or academic and industrial research.

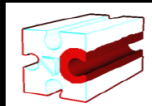


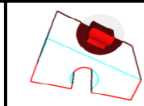




### **Building a Model**

Most of the single components to build fischertechnik applications are called blocks, which are made of polymers [49]. Few components (axles) are made of aluminum. Axles come in different lengths and are used for fixation or as rails. For connecting components together,

---

<sup>1</sup>[www.fischertechnik.de](http://www.fischertechnik.de), 27.11.2013

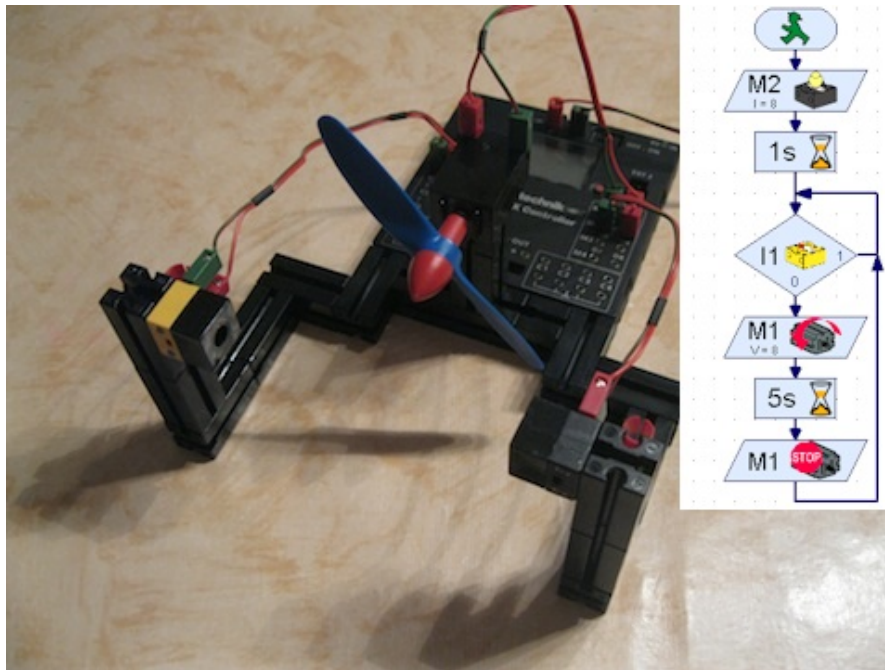
the blocks, girders and mechatronic components (motors, sensors) can have channels, pegs and holes. Figure 2.2 shows the connection principles for components.

	 channel	 hole	 axle	 peg
 channel				
 hole				
 axle				
 peg				

**Figure 2.2:** Connection rules for fischertechnik (green: connectable, red: not connectable)

### Transferring mechatronic Behavior to the Model

A built model, as in figure 2.3, is controlled by the “ROBO TX Controller“. This controller has a programmable processor, inputs and outputs. Sensors and actuators also provide the connection principles from figure 2.2. Reading and writing operations are transmitted over cables, which are connecting the “ROBO TX Controller“ with the sensors and actuators. The programming environment “ROBO Pro Software“ features visual programming. An example model with the corresponding visual program is depicted in figure 2.3.



**Figure 2.3:** Model of a hand dryer: The visual program starts by switching on the photo transistor M2. To adapt the photo transistor to the light the program waits for one second. Then it enters a loop and waits till something is triggering the photo sensor in front of the fan. On trigger, the fan blows for 5 seconds. After this the program enters the loop for reading data from the photo transistor again.

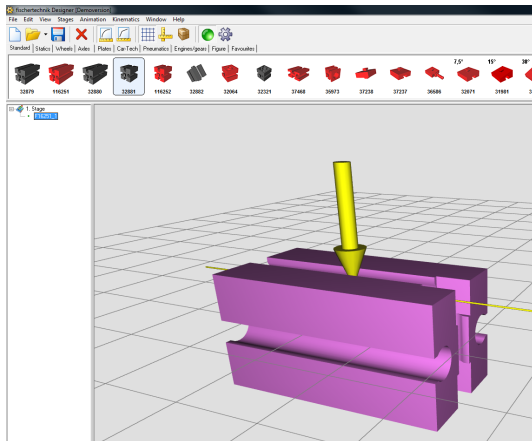
### Fischertechnik Designer

The commercial software *fischertechnik designer*<sup>2</sup> is a virtual building and simulation environment for fischertechnik parts. Models can be build with more than 500 different parts, which are kept up to date by regular updates. The spectrum of parts include static parts, motors, sensors and pneumatic hoses.

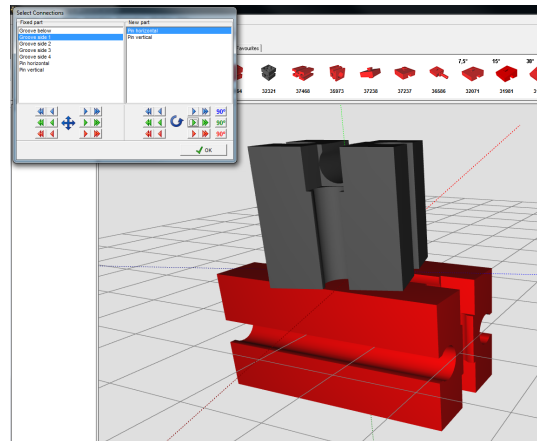
### Building a Model

Placing a part with *fischertechnik designer* is done by moving a part from the selection menu into the 3D window by Drag & Drop (see figure 2.4). The position of the new part is indicated only by an arrow. While dragging, the part is invisible. After releasing the mouse button, the part is connected to the other part. In case that the part was not placed correctly, as described in the connection rules (figure 2.2), a window opens which allows translation and rotation of the placed part (figure 2.5).

<sup>2</sup>[www.3dprofi.de](http://www.3dprofi.de), 27.11.2013



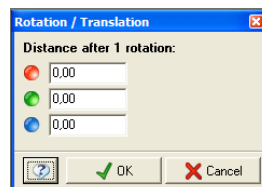
**Figure 2.4:** Attaching two parts together by Drag & Drop.



**Figure 2.5:** Changing position and rotation of the black part.

### Physical Behavior of a Model

The environment of fischertechnik designer features: different rotation bindings (gear rotation, rotation/translation binding), motor control, firm binding for parts placed on moving parts and collision detection. The physical consequences of these kinematic parts are not detected automatically, and therefore have to be set by the user. By the example of a gear wheel on a toothed shaft, figure 2.6 depicts how to set kinematic behavior.



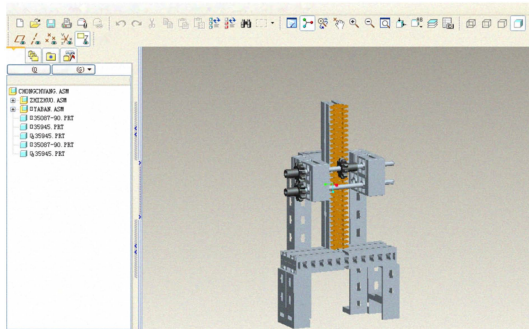
**Figure 2.6:** Setting physical behavior: First it has to be selected that these two parts join a rotation/translation relation. Then the driver and driven part are assigned. Finally the translation speed of the toothed shaft has to be set in relation to the rotation speed of the gear wheel.

When the physical behavior of the parts is set, the user can simulate a control flow with the “Logic Manager“. Similar to to “Robo Pro Software“ from the fischertechnik set, the “Logic Manager“ is a visual programming environment which features motor control, conditions and loops. The control of mechatronics stays in the virtual simulation, therefore no interface to actuators and sensors of a physical model is provided.

## Creo Parametric Software simulating Motion

The Creo Parametric<sup>3</sup> is a 3D computer aided design (CAD) modeling software. It features a 3D environment for modeling three-dimensional parts for mechanical engineering. Static models can be enriched with motion behavior which requires deeper knowledge of the software.

Xie described in [61] how to use Pro ENGINEER (a former version of Creo Parametric) for teaching the basics of mechanics. The example was to model virtual representations of fischertechnik components to simulate a gear wheel and a gear rack. The task for the student was to transfer rotational motion (gear wheel) to linear motion (gear rack). The paper does not tell about the complexity to define kinematic behavior. A video<sup>4</sup> explains how to transfer rotational motion to linear motion. The user has to set and adjust parameters which are required for executing the motion behavior (e.g. rotation center, rotation axis, object boundaries, linear motion for the gear rack...).



**Figure 2.7:** The simulating model [61].



**Figure 2.8:** The verification model [61].

The verification of the virtual model is done by building and simulating the model with real fischertechnik pieces (figure 2.7 and 2.8).

## Arduino

“Arduino is an open-source prototyping platform based on flexible, easy-to-use hardware and software.”<sup>5</sup> The published derivatives feature a programmable microcontroller, digital input/output

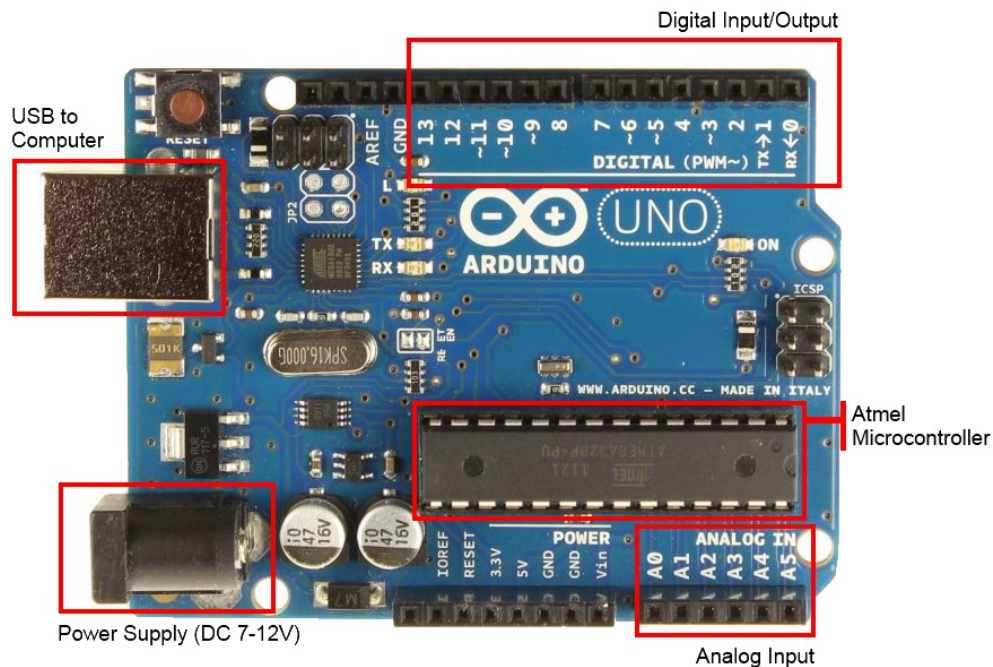
<sup>3</sup>[www.ptc.com](http://www.ptc.com), 27.11.2013

<sup>4</sup>[www.youtube.com/watch?v=7leLhYQj3qo](http://www.youtube.com/watch?v=7leLhYQj3qo), 2013.11.27

<sup>5</sup><http://arduino.cc>, 27.11.2013



pins and analog input/output pins. The programming language for the microcontroller is “Arduino programming language“, based on C/C++.



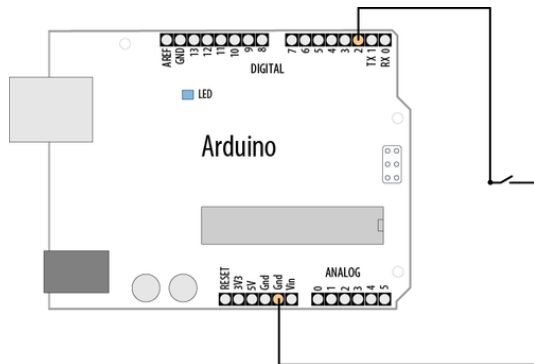
**Figure 2.9:** Arduino Uno circuit board with microcontroller (Atmel) and peripheral connections.

### Developing with Arduino

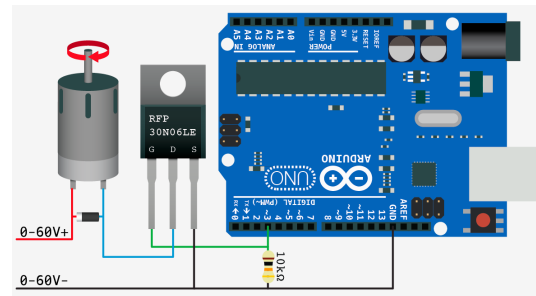
Developing mechatronic applications with Arduino requires basic skills in designing hardware-electronics and programming.

Figure 2.10 depicts how to connect a switch for reading digital signals. More complex in comparison is the layout in figure 2.11: The motor requires an external power source, and therefore also other electronics (transistor, diode and resistor). To change the behavior of the connected hardware the code for the microcontroller has to be reprogrammed. Another way is to develop a code and a hardware setup which is well enough implemented to allow changes of the behavior during run-time (e.g. by remote control [42]). But such a design could not provide the same features as reprogramming.

Writing code with the *Arduino programming language* can be avoided by using visual programming languages [14] [37].



**Figure 2.10:** A switch as input of the Arduino board. [42]



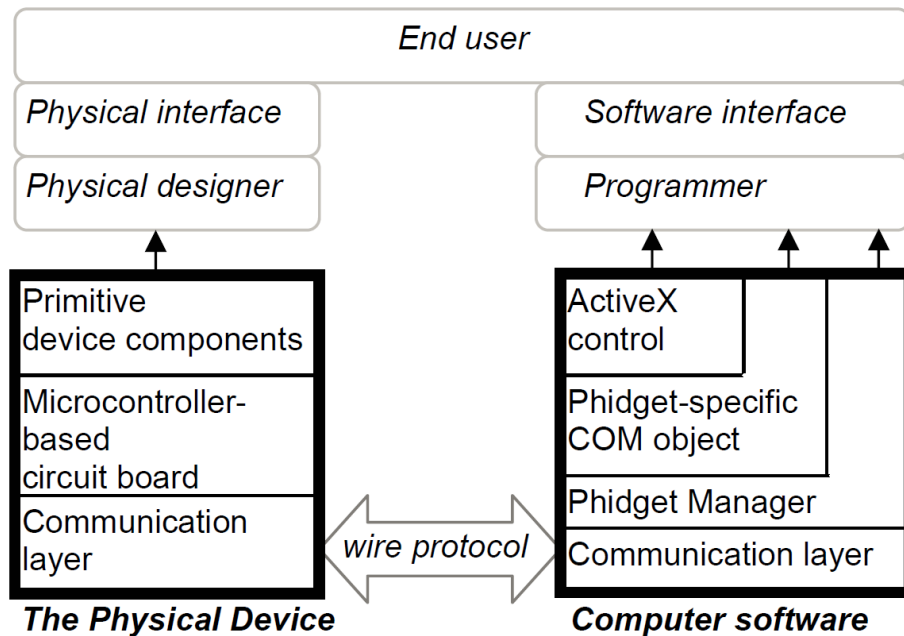
**Figure 2.11:** High power motor with external source, controlled by Arduino. [5]

## Phidgets

“Phidgets™, or physical widgets, are building blocks that assist in constructing physical user interfaces. The philosophy behind phidgets is: ... just as widgets make GUIs easy to develop, so could phidgets make the new generation of physical user interfaces easy to develop.“ [13] In this context, a physical user interface is a hardware device (actuator, sensor) that fulfills two goals: First, users should be able to design and control these hardware devices without working on low-level electronics. Second, programmers should be able to extend the physical interfaces.

## Plug & Play Architecture

Greenberg summed up the architecture of Phidgets: “... a phidget comprises a device, a software architecture for communication and connection management, a well-defined software API for device programming, a simulation capability, and an optional onscreen component for interacting with the device.“ [23]



**Figure 2.12:** Phidget Architecture [23]

The phidget architecture consists of the packages *physical device*, *wire protocol* and *computer software*.

**Physical Device** The end user who is in the role of a physical designer wants to use the physical devices by Plug & Play, without concerning electrical design. In Phidgets the primitive devices like motors, sensors or switches are extended by a microcontroller-based circuit board and a communication layer. The microcontroller-based circuit board translates the communication between primitive devices and the wire protocol.

**Wire Protocol** The wire protocol is a standard USB protocol which is implemented on the communication layer of the microcontroller and the Phidget Manager. A plugged-in phidget (primitive device + microcontroller) sends its name, a number and the state of the device over USB to the host computer.

**Computer Software** On the computer software side the Phidget Manager provides an API to the end programmer. The end programmer can look up from this API the phidgets name and index number. When a physical device is plugged or unplugged, events create a reference for the programmer to identify the device. Reading and writing operations on phidgets are instructed on the Phidget-specific COM<sup>6</sup> object. Each device corresponds to a specific COM object. The Phidget Manager handles the communication between a COM object and its physical device.

<sup>6</sup>[www.microsoft.com/com/default.aspx](http://www.microsoft.com/com/default.aspx), 27.11.2013

*Properties:*

```
DeviceType As String
IsAttached As Boolean
SerialNumber As Long
```

**Figure 2.13:** General COM interface definition (has to be implemented for every physical device).

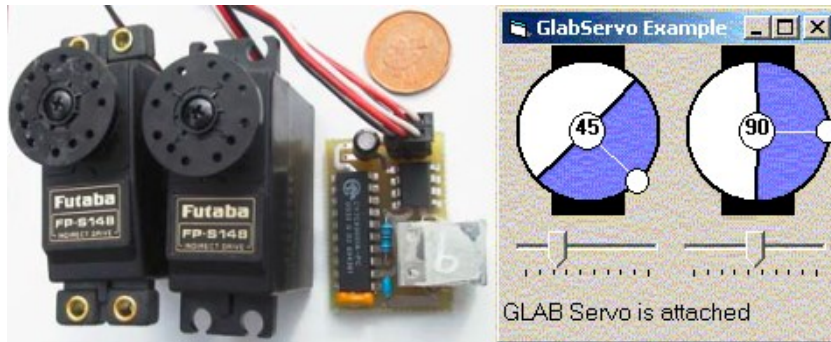
*Properties:*

```
MotorPosition(Index) as Integer
NumMotors as Integer
Events:
  OnPositionChanged (Index as Integer
                    Position as Integer)
```

**Figure 2.14:** Specific COM interface for a motor (implementation is optional).

At last, the *ActiveX control*<sup>7</sup> visualizes the phidget and its status on screen (Example in Figure 2.15). If a physical device is not present, a COM object can be generated for simulation. This simulated COM object also serves for visualizing via ActiveX.

### Physical device and Simulation



**Figure 2.15:** Servodrives, as physical device (left) and visual representation (right). [23]

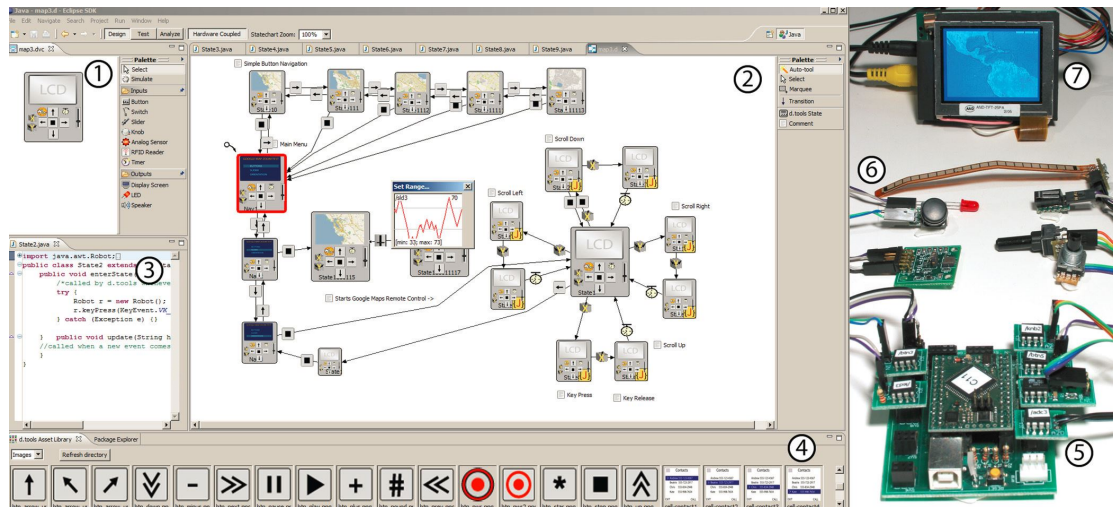
By the example of two servo drives, figure 2.15 depicts the physical device on the left, and the visualized ActiveX control on the right. A rotation on a physical servo drive lets the connected microcontroller instantiate and send a COM object, which is then received by the Phidget Manager and further processed to a visual representation in the ActiveX control.

If a physical device is unplugged during operation, the Phidgets environment can replace immediately the “unplugged“ COM object by a simulated one.

### d.tools

d.tools is a rapid prototyping toolkit for user interfaces with a focus on the design part. The intention is to quickly produce many prototypes in an iterative-design-centered approach. Barry observes: “the companies that want to see the most models in the least time are the most design-sensitive; the companies that want that one perfect model are the least design sensitive.“ [51]

<sup>7</sup>[www.microsoft.com/security/resources/activex-what-is.aspx](http://www.microsoft.com/security/resources/activex-what-is.aspx), 27.11.2013

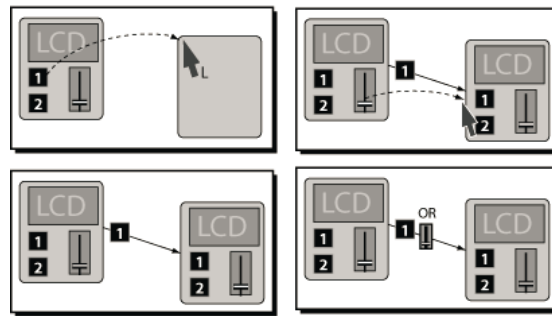


**Figure 2.16:** d.tools, visual and physical representation of a prototype. (1) device designer; (2) statechart editor; (3) source-code editor; (4) image browser; (5) hardware interface; (6) hardware inputs; (7). support for small LCD screens. [26]

Figure 2.16 illustrates the authoring environment of d.tools and hardware devices. With the visual, statechart-based prototyping model, the authors intent to provide a low threshold for early stage prototyping. The source-code editor augments the features of the statechart editor with textual programming. For software developers, the control of the hardware (physical interfaces) via the source-code editor is eased by providing library components. These library components shield away the low-level electronics from the user. Furthermore the physical user interfaces in d.tools provide extensible architecture: first, the interface between hardware and PC; second, the communication level between hardware devices; and third, the electronics on the circuit level. To analyze the prototyping process, the d.tools environment implements recording of the users progress with video and log files.

### Visual Design Tool

Usually the designer starts with plugging the hardware devices to the PC. Then the d.tools authoring environment recognizes the new devices on the USB port and creates virtual representations of the devices in the authoring environment. Featured devices are buttons, switches, sliders, knobs, RFID readers, LCD screens, LEDs and speakers. If a hardware device is not available, the virtual representation can be selected by Drag & Drop from the device editor's palette. Then the designer adds functionality to the devices in the statechart editor.



**Figure 2.17:** Statechart design with d.tools [26]

Figure 2.17 depicts an implementation example of a visual statechart formalism to control the behavior of virtual or physical devices based on digital data. In this example, a Liquid Crystal Display (LCD) device is either in state 1 (ON) or state 2 (OFF). The goal is to set the state to 1 after the next transition step. So any possible state of the LCD device has to change to 1 in the transition step (see top-right part). Bottom-right of figure 2.17 visualizes this transition with an OR symbol. Additionally the d.tools authoring tool provides timer for controlling the time of state changes.

Analog data from sensors (e.g. accelerometer) has continuous characteristics, but states are representing discrete data. Therefore the designer can assign user-defined thresholds of analog data (which spectrum can be visualized in the authoring tool) to states.

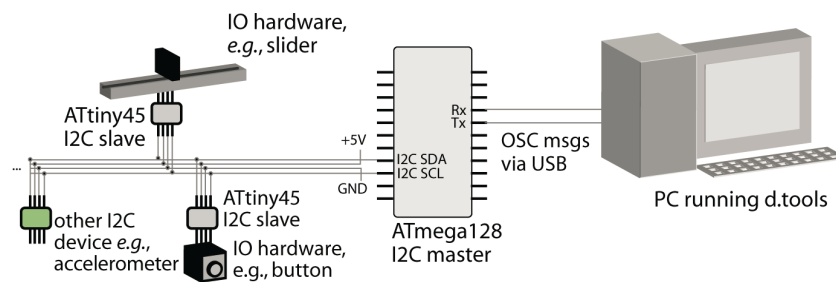
If the virtual devices do not fulfill the functionality of the hardware device, the Java code that represents the functionality of a device can be edited.

### The Prototyping Process with Video Recording

In d.tools, a video recording feature enhances the prototyping process. This feature is present in test mode, where parallel to the design process, the captured video is enriched with log data. Since the design process on the statecharts is also recorded on a data level, single video segments which correspond to statechart actions can be analyzed afterwards.

### System Architecture

The hardware devices feature Plug & Play, which is operated by cooperation of I<sup>2</sup>C protocol [30], hardware devices, a microcontroller, Open Sound Protocol (OSC) [60], Universal Serial Bus (USB) and a PC running d.tools software (see figure 2.18).



**Figure 2.18:** Phidget Architecture [26]

At the **hardware side** the master microcontroller, an ATmega128<sup>8</sup>, implements an I<sup>2</sup>C protocol that allows to plug multiple hardware devices on the I<sup>2</sup>C bus. The devices have to implement I<sup>2</sup>C slave protocols for being visible and controllable by the master microcontroller. Every hardware device consists of the basic I/O device (e.g. slider) and a microcontroller (ATtiny45) that implements the slave protocol. By design, the plugs of the hardware devices only allow correct plugging. Audio and video devices (LCD, speaker) are supported by the PC, therefore they are not addressed by the master microcontroller. Communication between the microcontroller and the PC is done by the OpenSoundProtocol (OSC).

The d.tools **software** on the PC side is written in Java JDK 5 [47] using Eclipse IDE<sup>9</sup>. The visual editors of the authoring environment are provided by the Eclipse Graphical Editing Framework (GEF). The video viewer is programmed in C#, and the synchronization between a video and its corresponding statechart is done by XML over UDP.

## Phybots

With Phybots [34] the authors provide a rapid prototyping environment for locomotive applications. It is argued that in rapid prototyping environments for physical user interfaces, only thin wrappers around the hardware devices are provided. But a complex locomotive application would require solutions to set the control behavior of sensors and actuators. The authors note also that in the field of robotics, solutions describe mostly specific robotic applications which require expensive sensors and actuators.

Therefore Phybots "... enable the rapid prototyping of user experiences with locomotive robotic things." [34] "Robotic things" outlines the support of Ikimo [15], LEGO Mindstorms NXT<sup>10</sup>, iRobot Roomba and iRobot Create<sup>11</sup>.

<sup>8</sup>www.atmel.com/Images/doc2467.pdf, 27.11.2013

<sup>9</sup>www.eclipse.org, 27.11.2013

<sup>10</sup>www.lego.com, 27.11.2013

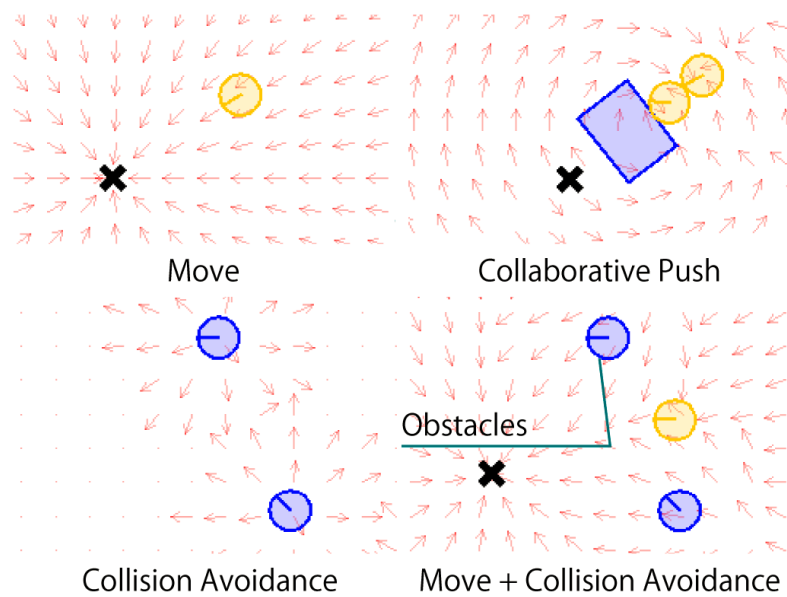
<sup>11</sup>www.irobot.com, 27.11.2013

## Prototyping Constraints

Due to the focus on robotics, the Phybots environment consists of certain hardware and software. The hardware is limited. A webcam, a robot, visual markers for obstacle detection and a PC underlines that the environment is developed for moving the robot automatically through a room. Also the software tool features APIs for 2D locomotion.

## Software Environment

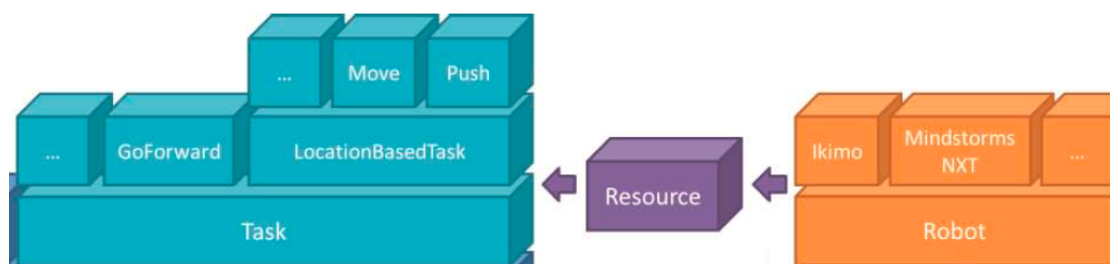
The focus lies on users with programming skills. A software API is provided that supports localization and navigation of a robot. With use of vector fields (figure 2.19) robots can navigate, push objects or avoid obstacles. For example The API for *move* depends on a vectorfield where all vectors point to the destination.



**Figure 2.19:** Vector fields for navigation. [36]

The software environment provides software services which represent tasks like “move“ or “push“. The capabilities of a single robot (e.g. wheels for driving and steering) are called resources. The tasks for complex robotic controls are using these basic resources. This works for every supported robot. Depending on the robot, resources have to be implemented differently to provide consistent interfaces for tasks. With this abstraction, tasks can be assigned to different robots.





**Figure 2.20:** Hierarchical software abstraction model [36]

**Sequential operations** are provided by the data structure *workflow*. It is a kind of directed graph where abstracted tasks denote nodes. Nodes for timeouts are also available. Workflows can be saved for later re-runs. The status of the robot and services are logged for **debugging**. [34]

### LEGO Mindstorms NXT

LEGO Mindstorms NXT, produced by The LEGO Group<sup>12</sup>, is a product series for creating programmable robots. In 1985, the Lego Company began sponsoring the MIT MediaLab<sup>13</sup>. This resulted in the development of the first programmable brick by MediaLab researchers. Nowadays the programmable brick is continuously developed and improved by the LEGO group (EV3, the next mindstorm platform is planned to be released in fall 2013). Due to its complex possibilities LEGO Mindstorms NXT is favored by a wide range of users (e.g. fanpages with user projects<sup>1415</sup>, education<sup>16</sup>).

<sup>12</sup><http://education.lego.com>, 27.11.2013

<sup>13</sup>[www.media.mit.edu](http://www.media.mit.edu), 27.11.2013

<sup>14</sup>[www.thenxtstep.blogspot.co.at](http://www.thenxtstep.blogspot.co.at), 27.11.2013

<sup>15</sup>[www.mindstormsforum.de](http://www.mindstormsforum.de), 27.11.2013

<sup>16</sup>[www.lego.com](http://www.lego.com), 27.11.2013



**Figure 2.21:** A Lego Mindstorms example which can carry objects, can move and has sensors attached.

### **Building a Model**

The single Lego components are called bricks. They are made of the thermoplastic acrylonitrile butadiene styrene (ABS), polycarbonate (PC) or polyamide [10]. Also the NXT microcontroller is referred as a *programmable brick*. The standard bricks have studs and holes for connection. The diameter of a stud (4.8 mm) is the basic unit in Lego and defines also the rules for connection of Lego Technic components like axles or gear wheels.

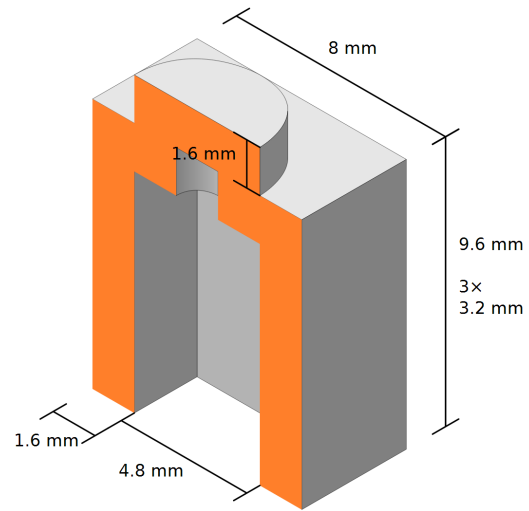
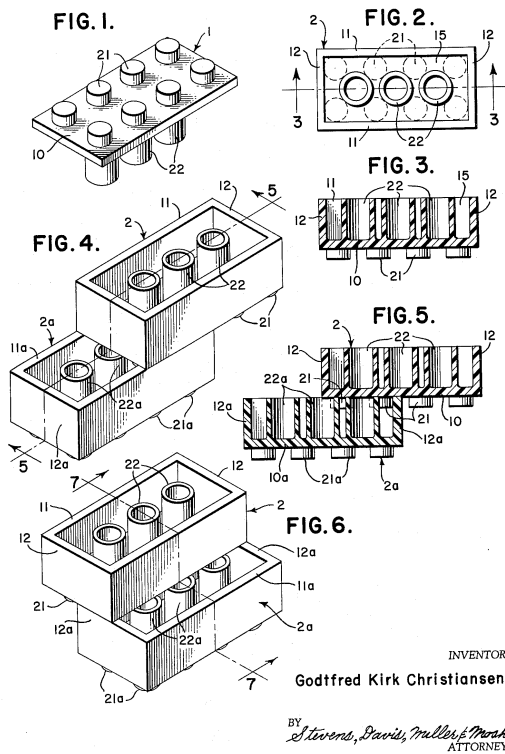
Oct. 24, 1961

G. K. CHRISTIANSEN  
TOY BUILDING BRICK

3,005,282



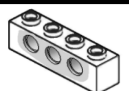





Filed July 28, 1958

2 Sheets-Sheet 1



**Figure 2.23:** Standard dimensions of a Lego brick. [43]

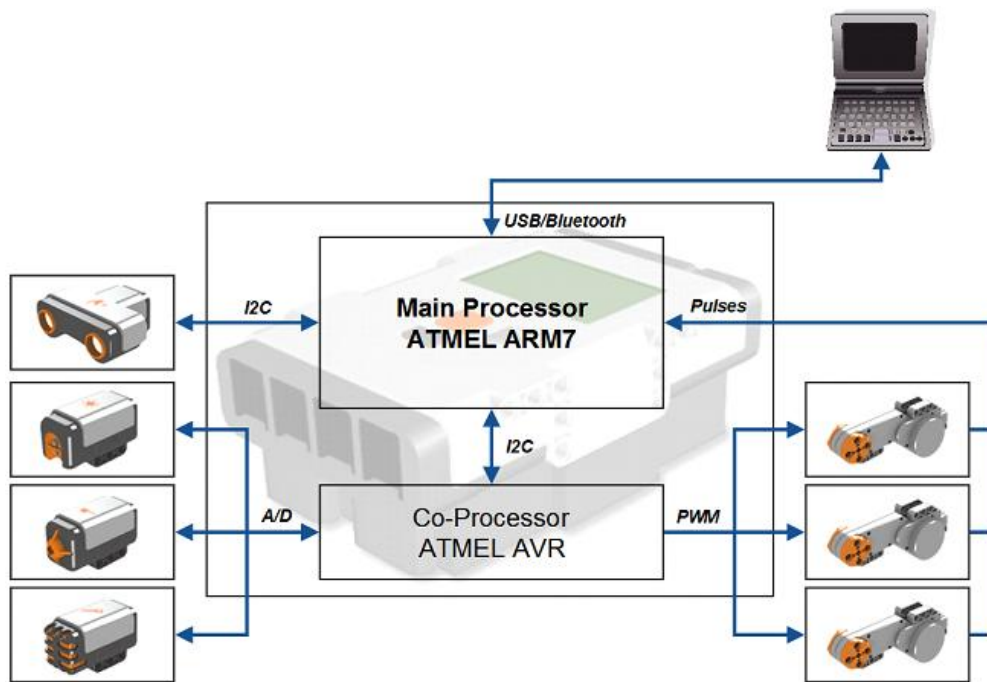
**Figure 2.22:** Patent for a Lego toy building brick. [9]

	 cross hole	 cross axle	 round hole	 round axle
 cross hole				
 cross axle				
 round hole				
 round axle				

**Figure 2.24:** Interfaces and connection rules of Lego Technic.

## System Architecture

The System Architecture consists of three main parts. At first a PC software to write programs for LEGO MINDSTORMS NXT robotic behavior. Second, the programmable brick that contains the microcontroller. And third, MINDSTORMS NXT devices (e.g. sensors, actuators) which take the commands from the main hardware for reading sensor data and executing actuator operations. The actuators and sensors of LEGO MINDSTORMS NXT provide the standard LEGO slots for plugging LEGO bricks on them. [39] and [48] provide knowledge and techniques to build mechatronic applications with LEGO.



**Figure 2.25:** System Architecture, LEGO Mindstorms NXT

In figure 2.25 the system architecture of the LEGO MINDSTORMS NXT<sup>17</sup> is visualized. The NXT main hardware device communicates with the PC via USB or Bluetooth. Inside the main hardware device are two processors. The main processor, an ATMELE ARM7<sup>18</sup>, is supported by a ATMELE AVR<sup>19</sup>, which serves the main processor as co-processor. The data communication with the PC is handled by the ATMELE ARM7 main processor. This processor communicates with its co-processor via an I2C bus. The main processor also provides an I2C connection to the ultrasonic sensor, while commanding rotation of motors and reading analog sensor data is executed by the co-processor. The motors are controlled with Pulse-width modulation(PWM) [57]. The data from the sensors is analog, so it has to be converted to digital

<sup>17</sup>[lejos – osek.sourceforge.net](http://lejos-osek.sourceforge.net), 27.11.2013

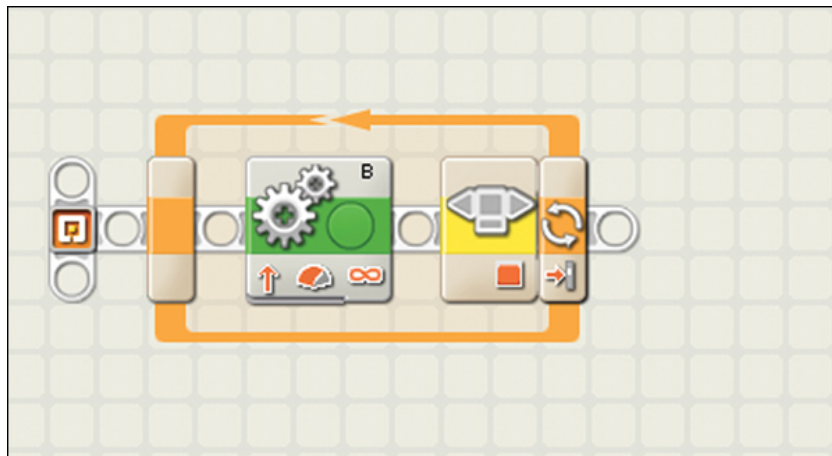
<sup>18</sup><http://arm.com>, 27.11.2013

<sup>19</sup>[www.atmel.com](http://www.atmel.com), 27.11.2013

format by an analog-to-digital converter [21]. Devices from different companies which connect to the NXT main hardware can be found at [www.generationrobots.de](http://www.generationrobots.de).<sup>20</sup>

## Software

**NXT-G** LEGO MINDSTORMS NXT is delivered with the programming environment NXT-G. It follows the visual programming approach [33]. The NXT-G environment is limited in its size of features. It is not possible to debug the code and the mathematics is limited by basic arithmetic operations. Therefore users often take a third-party programming environment (for example the Microsoft Robotics Studio 2.3).



**Figure 2.26:** NXT-G Visual code example. The loop (orange) iterates till a button (yellow) is pressed. While no button is pressed the motor (green) is running. [3]

**Virtual Simulation** The Lego Digital Designer <sup>21</sup> is a virtual construction environment for Lego applications in 3D. Building is done by dragging and dropping bricks from a selection menu to the 3D space. Applications are static, and therefore animation is not supported.

Two master thesis from the Universität Paderborn are describing the Lego Mindstorms Simulator (LMS). It takes leJOS<sup>22</sup> code (leJOS is a software for programming mindstorms applications with java) and simulates a mindstorms robot in a 3D environment. The 3D environment where the robot operates is described by a XML document, and it has no GUI editor for building 3D environments.

<sup>20</sup>[www.generationrobots.de](http://www.generationrobots.de), 27.11.2013

<sup>21</sup><http://ldd.lego.com>, 27.11.2013

<sup>22</sup><http://lejos.sourceforge.net>, 27.11.2013

## **Microsoft Robotics Developer Studio**

In the magazine Scientific American Bill Gates stated: “I can envision a future in which robotic devices will become a nearly ubiquitous part of our day-to-day lives.” [6] He also argued then that the robotic industry is lacking standardization in terms of operating system, hardware or programming language. Consistent with Bill Gates request’ for standardization, Microsoft developed a software environment for programming robots named Microsoft Robotics Developer Studio (MRDS)<sup>23</sup>. Several robotic systems are supported (e.g.: LEGO MINDSTORMS NXT, Parallax<sup>24</sup>, KUKA<sup>25</sup>). Besides, advanced users can control yet unsupported robots by creating a new hardware interface.

## **Software Architecture and Services of MRDS**

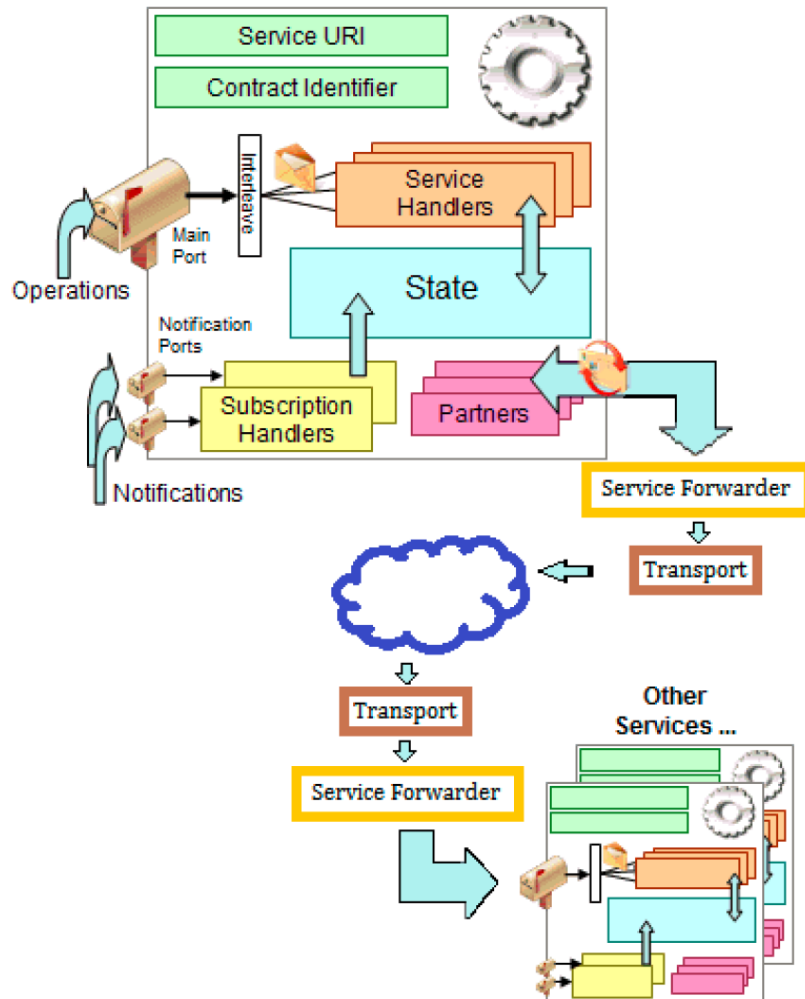
The software architecture of MRDS is a service oriented architecture, where services represent basic features of robots which can be orchestrated for complex applications.

---

<sup>23</sup> [www.microsoft.com/robotics](http://www.microsoft.com/robotics), 27.11.2013

<sup>24</sup> [www.parallax.com](http://www.parallax.com), 27.11.2013

<sup>25</sup> [www.kuka-robotics.com](http://www.kuka-robotics.com), 27.11.2013

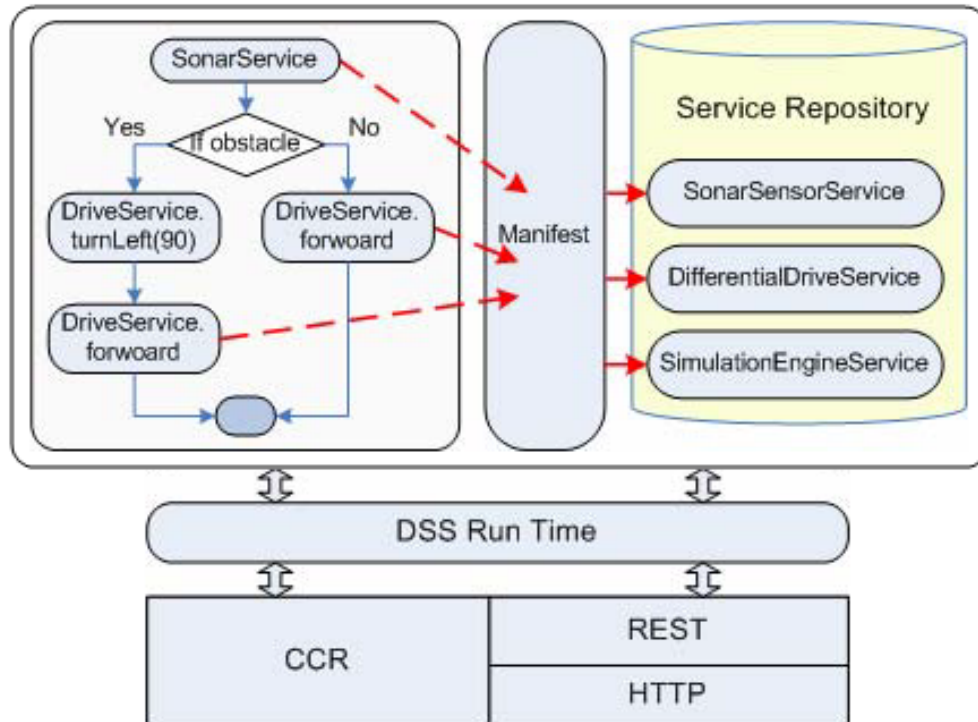


**Figure 2.27:** A service with its seven elements: Main Port, Service Handlers, Event Notifications, Service State, Service Patterns, Contract Identifier and Universal Resource Identifier. [8]

The interface of a **service** is the *Main Port*. It can be connected to other services, actuators or sensors. In figure 2.27 the *Main Port* is depicted as one out of seven service components. Depending on the received message type on the *Main Port*, the correspondent *Service Handler* operates (read/write) on the *Service State*. The *Service State* contains several state information like speed, distance or color. If the state is subscribed to other services, the *Subscription Handlers* inform other states about the current state change or state request. These subscribed services receive the message also on the *Main Port*. The *Service Partners* define the “wiring” between services and therefore allow complex orchestration of services. For uniquely identification, services have a dynamical *Universal Resource Identifier (URI)* and a *Contract Identifier*. The *Contract Identifier* is unique for each defined service, provided that only one instance of a service exists. To distinguish two instances of the same service, the dynamical *Universal Re-*

*source Identifier (URI)* provides a unique URI for each service instance. [8]

The MRDS **software architecture**, depicted in figure 2.28, is based on the .NET [41] framework which allows to use any of .NET supported languages like C# or Visual Basic .NET (VB.NET).



**Figure 2.28:** Robotics Studio Architecture [54]

The Concurrency and Coordination Runtime (CCR) is a .NET library which allows asynchronous operations on the robot. This means that the robot can perform two or more operations in parallel (for example reading a distance sensor input while moving forward). The programming model of the CCR supports this parallelism without the use of complex principles like manual multi-threading, semaphores or mutual exclusions. Reading/Writing operations on sensors/actuators are called messages, which are transferred on so-called *PortSets*. When a *PortSet* gets a message from the Main Port of a service, the receiver transmits it to the *arbiter* which dispatches it to the multi-threading pool (Dispatcher). From this multi-threading pool the messages are dispatched to sensors, actuators or services. [8].

The Decentralized Software Service (DSS) is built on top of the CCR, which loads services and manages the communication between services via the *Service Forwarder*. The communication is based on Representational State Transfer (REST) [17] by using Hypertext Transfer Protocol (HTTP) [16] as a communication protocol and Extensible Markup Language (XML)<sup>26</sup>. As transport protocol, Simple Object Access Protocol (SOAP)<sup>27</sup> is used. It is extended with the

<sup>26</sup>www.w3.org/XML, 27.11.2013

<sup>27</sup>www.w3.org/TR/soap, 27.11.2013



feature of subscriptions. Subscriptions allows a service A to be notified by another service B if the state of service B changes.

The robotic behavior is programmed with one of the supported language C# or Visual Programming Language (VPL). The programmed implementation uses services for orchestrating (i.e. a workflow of services) the robotic behavior. With the use of generic services the user can run its coded application on different robots and in simulation mode. But a generic service is abstract and may use different services from different robots, which are listed in the service repository. So “lower level“ services are translated to generic services before a simulation or “real robot operation“ starts. The service synonyms and translation information are described in “manifest“ XML configuration files. [54] It allows programmers to use existing services with new hardware.

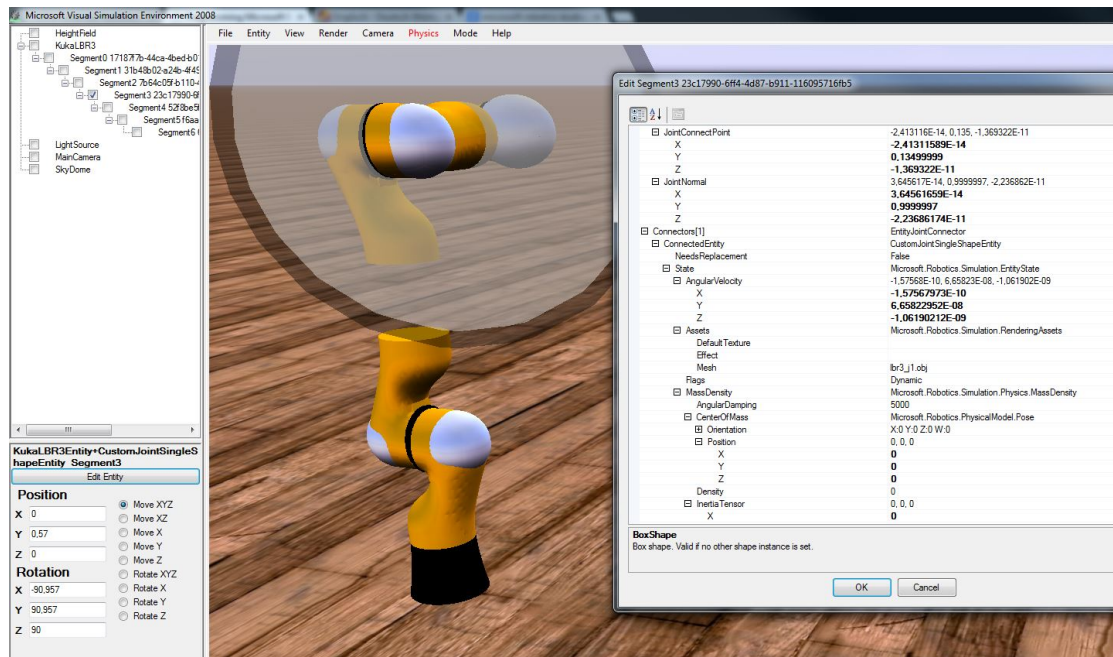
### **Visual Simulation Environment**

With the Visual Simulation Environment (VSE) users can develop robot applications without hardware. It uses DirectX 9 and Microsoft XNA<sup>28</sup> for 3D rendering in real-time. For simulating physical behavior VSE includes the PhysX engine<sup>29</sup>. MSRS provides several prearranged simulations (e.g.: KUKA LBR3 Arm simulation, LEGO NXT Tribot simulation). In the simulation a 3D object is described as an entity. Entities can be arranged hierarchically, which is necessary for simulating robotic behavior. The simulator offers an edit mode, where entities can be loaded and edited in terms of behavior, position, rotation and parent/child relationship. Entities which are sensors or actuators can then be coupled with a service. The textbox in figure 2.29 shows the complex possibilities of arranging a joint to another.

---

<sup>28</sup>[msdn.microsoft.com](http://msdn.microsoft.com), 27.11.2013

<sup>29</sup><http://www.geforce.com/hardware/technology/physx>, 27.11.2013



**Figure 2.29:** Editing a robotic arm in MSRS: In the left pane a segment, embedded in a hierarchical structure, is selected. The position and rotation of the segment can be changed arbitrarily. In the 3D view the selected segment is highlighted with a transparent sphere. The right pane displays segments which have multiple detailed options to adjust. Connection points to other segments can be defined. For example textures of objects can be changed or physical mass information is editable. [45]

## Programming

The Visual Programming Language (VPL) is based on statecharts and works therefore similar to the LEGO MINDSTORMS NXT-G environment. MSRS is embedded in the .NET framework, so software for robots can also be created with supported programming languages like C#.

## 2.4 Comparison and Summary of Existing Approaches

This section evaluates the prototyping environments (section 2.3) on the basis of the evaluation criteria (section 2.2). First, an evaluation overview of criteria on prototyping environments is given in table 2.30. Second, the detailed argumentation for each prototyping environment is given afterwards in section 2.4. At last in the Research Conclusion 2.4, the best matching prototyping environments are named, and also an optimal prototyping environment is sketched. This “optimal“ prototyping environment is a pick and choose from the evaluation criteria, and gives a wishlist for a prototyping environment.

## Overview

		Prototyping Environments												
		Performance Rating												
		Good	Average	Poor	Not supported	Fischertechnik	Fischertechnik Designer	Creo Parametric Software	Arduino	Phidgets	df tools	Phybots	LEGO M. NXT	MRDS
Category	Criteria													
Prototyping	Prototyping process	Good	Average	Good	Good	Good	Good	Good	Good	Good	Good	Good	Good	Good
	Prototyping of mechatronic systems	Good	Good	Poor	Good	Good	Good	Good	Good	Good	Good	Good	Good	Good
	Virtual prototyping	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported
	Physical prototyping	Good	Good	Good	Good	Good	Good	Good	Good	Good	Good	Good	Good	Good
	Visual authoring	Good	Good	Poor	Good	Good	Good	Good	Good	Good	Good	Good	Good	Good
	3D modelling	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported
	3D control	Not supported	Not supported	Poor	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported	Poor
	2D control	Good	Good	Good	Good	Good	Good	Good	Good	Good	Good	Good	Good	Good
	Feedback system (debugging)	Good	Good	Good	Good	Good	Good	Good	Good	Good	Good	Good	Good	Good
	Real-time behaviour	Good	Good	Good	Good	Good	Good	Good	Good	Good	Good	Good	Good	Good
	Usability	Good	Good	Good	Good	Good	Good	Good	Good	Good	Good	Good	Good	Good
	Understandability	Good	Good	Good	Good	Good	Good	Good	Good	Good	Good	Good	Good	Good
	Operating options	Good	Good	Good	Good	Good	Good	Good	Good	Good	Good	Good	Good	Good
	Attractiveness	Good	Good	Good	Good	Good	Good	Good	Good	Good	Good	Good	Good	Good
Technical	Interfaces of the System Architecture	Good	Good	Good	Good	Good	Good	Good	Good	Good	Good	Good	Good	Good
	Hardware	Good	Good	Good	Good	Good	Good	Good	Good	Good	Good	Good	Good	Good
	Complexity	low	Good	high	high	Good	Good	Good	Good	Good	Good	Good	Good	Good
	Extendibility	Good	Good	Good	Good	Good	Good	Good	Good	Good	Good	Good	Good	Good
	Portability	Good	Good	Good	Good	Good	Good	Good	Good	Good	Good	Good	Good	Good
	Software	Good	Good	Good	Good	Good	Good	Good	Good	Good	Good	Good	Good	Good
Economic	Price (€)	400	120	200/Y	80	230	80	50	500	400				
	Availability	Good	Good	Good	Good	Good	Good	Good	Good	Good	Good	Good	Good	Good
	Popularity (downloads/citations)	Good	Good	Good	Good	Good	Good	Good	Good	Good	Good	Good	Good	Good
	Support	Good	Good	Good	Good	Good	Good	Good	Good	Good	Good	Good	Good	Good

Figure 2.30: Evaluation of Prototyping Environments

## Detailed Evaluation

### Fischertechnik

The Fischertechnik construction kit, including the ROBO TX controller and software, features support for rapid prototyping for mechatronic applications very well. Models can be build and tested step-wise. But it lacks of a virtual 3D modeling and simulation environment. Therefore the user has to know which 2D representation matches to the analogical model parts. During

execution, the programs (“flowcharts“) coded with its visual programming language are giving visual feedback by marking the current state (in online mode). The purpose of the Fischertechnik environment is made clear to the user. Due to its focus on children, the environment is usability friendly and operating options are clear.

The cables from ROBO TX controller to actuators/sensors have to be connected correctly in terms of positive/negative power supply.  $I^2C$  devices from third party manufacturers can be connected and used with the software. New plugged-in devices are not automatically recognized by the software. Hardware devices from Fischertechnik are not complex, because “data“ and “instructions“ are mostly handled with the level of voltage. Extending the system with third party devices or software is possible, but requires skills in software and hardware design. Both, the hardware and software is well portable.

A set which includes the software, the hardware controller, actuators, sensors, an accumulator and static building pieces costs about 400€<sup>30</sup>. It is available worldwide in various stores. Since it is very popular (e.g schools, universities, fanpages on the web), it is also well supported by Fischertechnik, third party manufactures (sensors), and users.

### **Fischertechnik Designer**

The fischertechnik designer features only 3D simulation for Fischertechnik models. Since the Plug & Play does not work as expected and the physical behaviors have to be set by the user, the prototyping process takes too much time with settings that goes beyond rapid prototyping. The use of the logic manager (visual programming tool) requires also a predefinition of physical behavior by the user. The purpose of the software is not obvious in the first moment and the usability is not intuitive. The 3D environment features rendering quality options for different computer hardware. Also the 3D models of Fischertechnik components are well shaded and have a high polygon count.

The software can not be extended by the user and features also no interface to hardware like motors or sensors. In the simulation, use of sensors is limited. It has more features than ROBO TX software, and is therefore more complex.

A free demo version is available, and the price of the full version of fischertechnik designer is about 120€. It has a fanbase (about 80 designs are listed at [www.ftcommunity.de](http://www.ftcommunity.de)<sup>31</sup>) and is also used in schools. The software is regularly supported with updates.

### **Creo Parametric Software**

The software features design of models which can be used for manufacturing. The complexity of usability hinders virtual rapid prototyping for mechatronic applications. A interface to control a real prototype is not provided. The visual authoring environment is powerful but requires a deep knowledge of the software. Dependency of motion of more than one object is not recognized automatically and has to be defined in detail. The software can be purchased by download and the license costs about 200€ per year.

---

<sup>30</sup>Estimated with prices at: [www.amazon.com](http://www.amazon.com), 27.11.2013

<sup>31</sup>[www.ftcommunity.de](http://www.ftcommunity.de), 27.11.2013

## **Arduino**

Prototyping with Arduino requires programming skills. But due to its open design, a wide range of applications can be developed. For mechatronic applications, actuators and sensors need to be purchased from third party publishers. Also the first setup, calibration, configuration and data communication has to be developed by the user. But the user may find already coded parts in literature or the internet. The software from Arduino features only textual programming, but third party software allows visual programming. Programs can be coded to listen on the input and display status to the user. Also the feature of interactive status changes can be coded. Hardware status can therefore be read and set in real time. Users without programming experience may not understand the purpose and benefits of Arduino. But if skills in programming are present, the prototyping process can be executed accordingly. The programming environment features syntax highlighting but no auto completion.

The Arduino board is connected to the PC via USB, where compiled programs are transmitted to the microcontroller on the circuit board. The complexity of Arduinos input and output pins depends on the used actuators or sensors. Since a wide field of interfaces can be used, the rating is good. The hardware complexity is high, but it can be extended. The same applies for the programming environment.

The Arduino Starter Kit costs about 80€ and includes: Arduino Uno, power supply, USB cable, software (free to download), wires, sensors, actuators and further electronic components.

## **Phidgets**

The Phidgets tool was given to students with programming skills with the task of developing creative applications. Then they had to give feedback about the development process. They reported that most of the time was spent on designing kinematic functions. Programming took less time and development on low level electronics was done just by a few students. [23] Therefore prototyping of mechatronic systems works quite well with Phidgets. It features simulation of devices that are unplugged or not available, but the ActiveX representations in the software environment are two-dimensional and require additional programming. Visual authoring is not provided, programming skills are required. Since changes on devices are transmitted back to the software environment (for updating the 2D ActiveX representation), the hardware and software are operating parallel in real time. Featured devices are designed for correct plugging. Hardware knowledge is required, especially when extending the current system with new devices.

Phidgets supports a wide range of programming libraries and hardware devices. A starter kit starts at 230€.<sup>32</sup>

## **d.tools**

Feedback for the d.tools environment was gathered in three evaluations.

First, in a 90 minutes long controlled laboratory study, tasks were given to students who were mostly in design related field of studies. The automatic recognition of devices and the authoring with visual statecharts were perceived intuitively. But changing the properties of statecharts was

---

<sup>32</sup>[www.phidgets.com](http://www.phidgets.com), 27.11.2013

less intuitive. After some time of learning the environment students were able to spend their time on the interface design. Afterwards, a survey which asked for the opinion of students resulted in: mean value  $\mu = 4.3$  (out of 5) for time savings in prototyping;  $\mu = 4.6$  in usability testing; and  $\mu = 4.25$  in helping to understand the user experience during the design process. But it was criticized that no software simulation exists which works without attached hardware. It was also claimed by the students that sensor ranges had to be set textually instead of graphically. [26]

In a second evaluation the capability of expressions of the visual authoring environment (statecharts) was evaluated. Therefore the interface functions of three electronic devices (MP3 player, digital camera, PDA) were recreated with statecharts in d.tools. The outcome was, that on the screen up to 50 states are visually understandable. In this evaluation, the design part took about 70% and implementation took about 30%. [26]

Third, students of a HCI (human computer interaction) course were free to choose d.tools for their final project. The purpose of the project was to design a tangible interface. Students that used d.tools were able to extend d.tools (sensor that was not part of the library). [26]

The software is free under the open source BSD license, well documented and regularly updated. Hardware for d.tools has to be purchased from third party developers (wiring hardware platform, arduino). At sourceforge, d.tools was downloaded over 3900 times since 2006. It was rated by just three users, which has no explanatory power. Since it has Arduino support, the price of 80€ is based on the Arduino starter kit.

### **Phybots**

Phybots is a robotics toolkit, where prototypes are limited to navigation of the robot, on the basis of camera detection and with help of visual markers. Mechatronic prototypes like a 2D plotter can hardly be developed. The software side requires programming skills and features no visual programming. For new robots, the software model is extendable by programmers. An Ikimo Robot Platform costs about 50€ and the Phybots software is free under GNU GPLv3. The software receives regular updates and the documentation on the webpage is japanese.

### **LEGO Mindstorms NXT**

Similar to Fischertechnik, LEGO Mindstorms NXT let enhance bricks with visual programming. Having a commercial product for children results in well intuitive usability. Virtual modeling and controlling in 3D is not supported (the LEGO Digital Designer features 3D modeling, but no interface to Mindstorms NXT). Programs have to be compiled, so no real-time behavior is supported. Feedback from programs can be realized by doing some workaround on virtual sensor components. But real debugging is not supported. Sensors and actuators which fit to the interface are available also from third party producers. But extending hardware at low level side is not supported. The programming software NXT G is closed and can not be extended, but it can be replaced by third party development environments. A Mindstorms roboter costs about 500€.

## **Microsoft Robotics Developer Studio**

The MRDS is a development environment, not a prototyping environment. Indeed, features like visual programming, virtual simulation, physics implementation or an interface to sensors and actuators are present. But it requires certain knowledge from the user to work with it. This knowledge contains service oriented architecture, XSD, limits of the physics engine and microsoft visual studio framework. MRDS provides only the software environment to control or simulate robots. Robots have to be purchased from third party companies. Building of virtual applications in 3D requires parameter setting of each component (e.g. position, rotation). The correct mechatronic behavior of components requires an assignment of the correct service and also parameter settings. When the configuration is finished, the control (navigation) of the robot is done by a usability friendly interface. The power of the software environment makes the usage and its operating options not clear to the user in the first moment.

On a technical point of view, MRDS with its service oriented architecture features extendability at hardware and software side. The complexity of hardware relies on the used third party devices. The development environment is free, and the price of 500€ includes the calculation for a LEGO Mindstorms hardware set. The community is quite active. But the underlying game engine XNA for developing 3D applications will not further be supported [50].

## **Research Conclusion**

A physical functional 2D pen plotter, developed in a prototyping process, can be achieved with:

- Fischertechnik
- Arduino (with use of actuators, sensor and static pieces)
- Phidgets
- d.tools
- LEGO Mindstorms NXT
- MRDS (with use of third party hardware)

A simulated (virtual) 2D pen plotter, developed in a prototyping process, can be achieved with:

- Fischertechnik Designer
- Creo Parametric Software
- MRDS

Therefore the Microsoft Robotics Developer Studio (MRDS) environment is the only tool which features both, physical and virtual prototyping for applications in 3D. But MRDS is rather more a development environment, than a rapid prototyping environment for mechatronic applications. The marking characteristics of the MRDS environment are: high complexity, integration

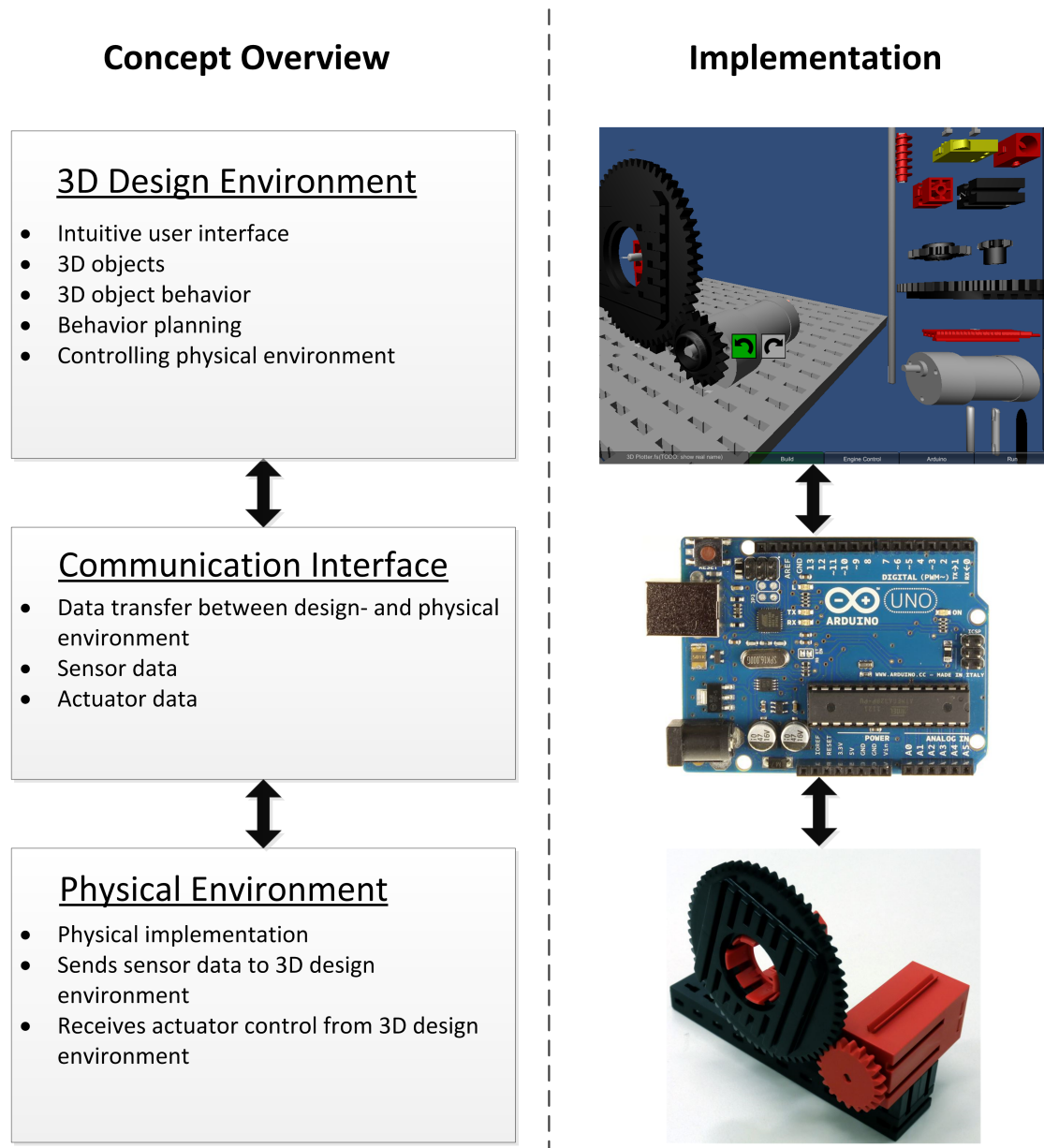
from different frameworks (Visual Studio, XNA, PhysX, VPL) and average ratings in the key criteria for prototyping. Furthermore it does not argue for MRDS that the game development environment XNA is phased out.



# Concept for a Mechatronic Design Environment

Since the presented environments are not fulfilling the criteria in the evaluation catalog, a design and implementation for a development environment for mechatronics in a rapid prototyping approach is proposed. The concept for a mechatronics design environment is given in section 3. Second, the design and implementation for this rapid prototyping environment is given.3.1.

### 3.1 Concept

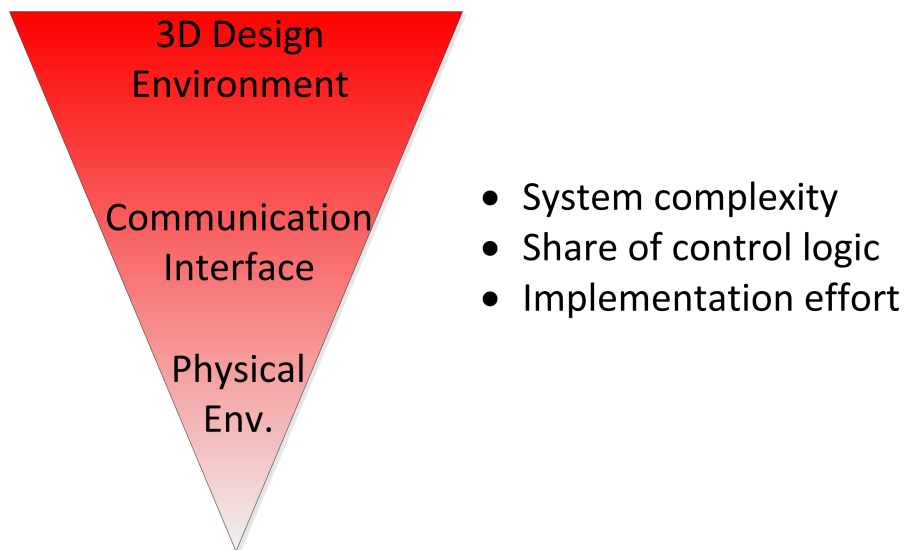


**Figure 3.1:** Main components of the concept and a concrete implementation for a mechatronic design environment

For this mechatronic design environment the goal is to develop prototypes in a virtual environment where no programming or electronics knowledge is required. The whole mechatronic

design environment consists of three blocks: First, a virtual 3D design environment. Second, a physical design environment. And third a communication interface which transmits data between the virtual and the physical environment.

Figure 3.2 depicts that the main knowledge and core functionality lies in the 3D design environment. It is assumed that a physical environment including sensors and actuators already exists. Although this physical environment may have its own control environment which would not be used. The communication interface passes signals between the PC and the physical environment. These signals are controlled from the 3D design environment where the most development effort is required.



**Figure 3.2:** Weighting of the main components (top red weights most) in terms of system complexity, control logic and implementation effort.

### 3D design environment

The 3D design environment is a virtual interaction environment where the user can prototypically build and control mechatronic applications.

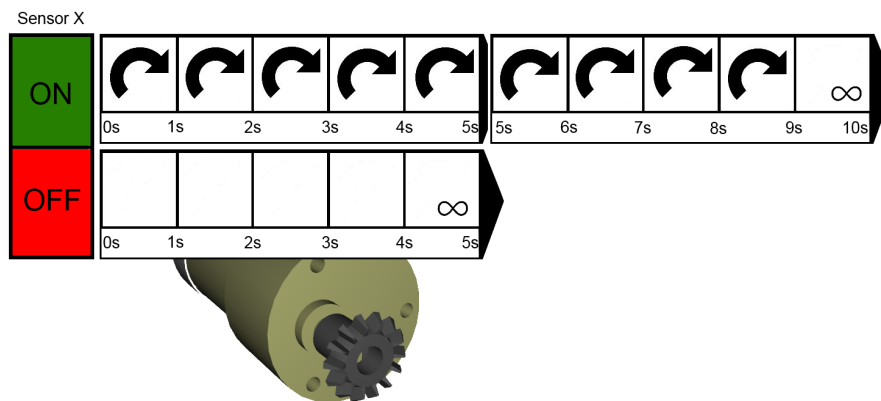
**Building with pieces:** It features pieces which are enriched with connection information based on connection rules (e.g. in figure 2.2 and 2.24). So it is possible to build any possible static three-dimensional application. Of course, the visual impression of the build application is defined (or limited) by the form, connection rules, polygon size and texturation of the single pieces.

**Building environment:** The user can navigate the virtual camera through the 3D design environment during the prototyping process. Built applications or prototypical fragments can be saved and loaded anytime. The building process is derived from a real construction environment like LEGO or Fischertechnik. A ground plate (the floor) with connection slots is present in the

bottom center of the environment. Choosable Pieces for building are located on the side, top or bottom of the viewing area (e.g. computer screen, 3D glasses, virtual reality headset). During navigation through the 3D environment, the pieces still stick to the defined section of the screen. This way the user can take a piece and move it to the ground plate or already placed pieces (by e.g. drag & drop with the mouse, hand arm gestures in virtual/augmented reality).

**Mechatronic pieces:** Actuators (e.g. Motors, LEDs), sensors (e.g. button, light sensor) and kinematic components (e.g. gear wheel, screw thread) are also pieces with connection rules and can therefore be connected with other pieces. The features of the mechatronic pieces are calculated after placing them, gear wheel transmissions, screw thread functionality, sensors). The source of motions are motors. For rapid prototyping motors are enriched with buttons for rotation. Immediately after a motor is placed these buttons are displayed and usable. Attached kinematic components do not require a setup by the user. They rotate or translate physically correct from start (also chains of gear wheels behave correct). Sensor states can be coupled to actuators behavior also in the 3D environment.

**Mechatronic design:** The prototypical mechatronic design enables the setup of a control flow for the mechatronic pieces where programming skills are not required. In the 3D environment timing panels next to the mechatronic pieces are displayed where the user can set actions for each defined timeframe. A timeframe is defined as a constant time span (e.g. one second). This way an actuator's status can be set for each second. By Drag & Drop sensors can be attached to actuators. Then the actuator's status is definable depending on the sensor status. Defined control flows can be started, paused and reset in the 3D view.



**Figure 3.3:** A timing panel for a motor depending on a sensor. In this configuration the motor rotates clockwise while the sensor is set to ON. Otherwise the motor is stopped.

### Communication interface

The defined control behavior of mechatronic prototypes built in the virtual 3D environment need to be transmitted to a physical implementation of the prototype. status of actuators is sent

from the 3D environment to the physical actuators, and sensor status sent by physical sensors is received by the 3D environment. Therefore a communication interface which passes through the signals to the right pins and ports is required. The mapping of virtual actuators and sensor to their corresponding physical actuators and sensors can be done in the virtual 3D environment.

### **Physical design environment**

It is assumed that the physical prototyping environment already exists (e.g. LEGO, Fischertechnik). Prototypes built in the 3D design environment can then be realized as physical models. Its sensors and actuators are then connected to the defined pins (set in the 3D environment) on the communication interface. When the control flow is started in the 3D environment, it is synchronously transmitted to the physical prototype. This way the virtual and physical prototype are running in parallel and behaving the same.

## **3.2 Design and Implementation**

### **Main Components of the proposed Environment**

**Unity game engine as Software Framework:** A suitable software framework which can be used for developing a virtual rapid prototyping is required. MRDS provides services for controlling robotic systems like fischertechnik or LEGO Mindstorms. Its game engine XNA, where the simulation environment for MRDS is implemented, could be used for developing a new simulation environment to fulfill the requirements from section 2.2. As mentioned before, future support of XNA is not provided. It also has no 3D editor integrated into the development environment. Loading assets like polygon models can only be done via code lines.<sup>1</sup> A game engine is requested which enhances rapid loading of polygon models and which makes these polygon models extendable with physical functions. Also GUI development and connection to external hardware has to be supported. A game engine that fulfills these features is the Unity game engine <sup>2</sup>.

**Arduino as Communication Hardware:** A direct connection of the PC to actuators and sensors of the construction is hardly possible. Somewhere in between must be space for changing physical connections (wire plugging) or providing power for actuators. The arduino board, described in section 2.3, provides these features to serve as an interface between a PC and the construction set.

**Fischertechnik as Construction Set:** LEGO Mindstorms and Fischertechnik are two possible construction sets. In the requested rapid prototyping environment the sensors, actuators and static pieces of the construction have to provide precise animation. Also the 3D polygon models of construction pieces are required to avoid 3D modeling. The polygon models of the fischertechnik designer<sup>3</sup> can be exported for the use in Unity. The community favors plotters

---

<sup>1</sup>[www.microsoft.com/en-us/download/details.aspx?id=23714](http://www.microsoft.com/en-us/download/details.aspx?id=23714), 27.11.2013

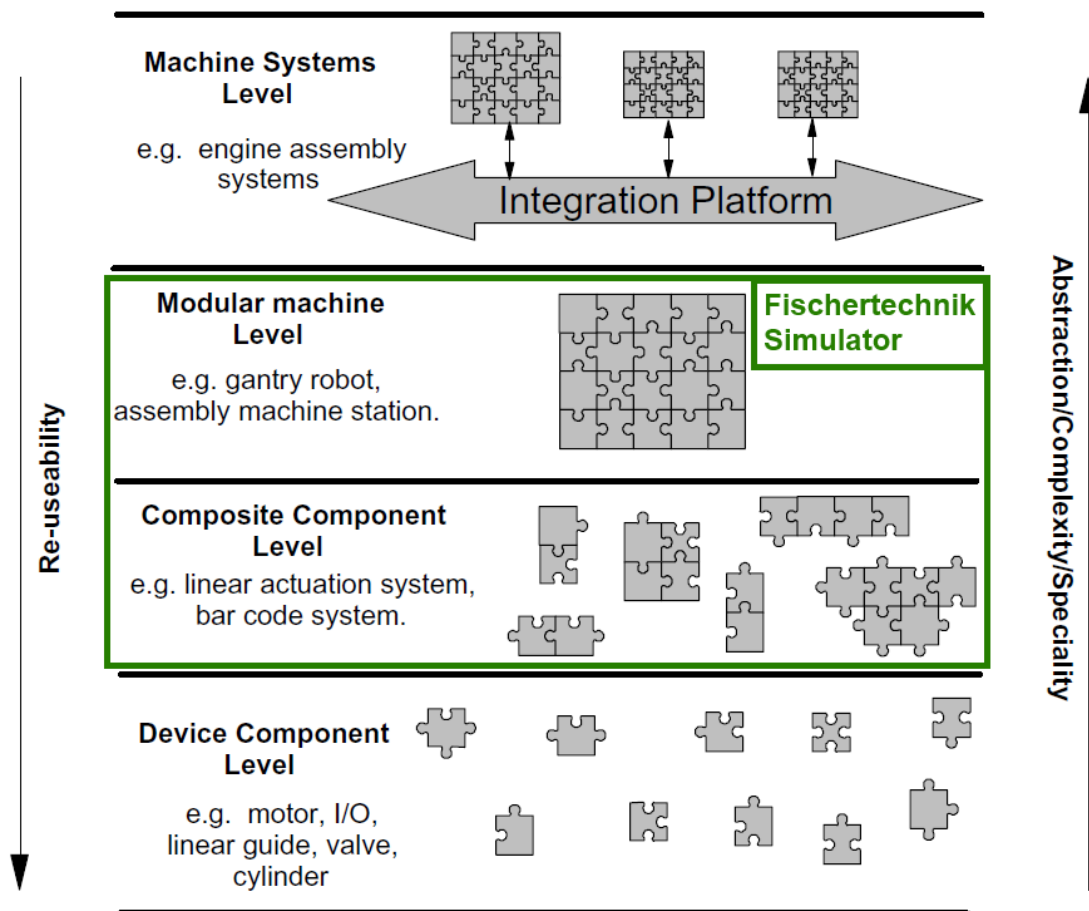
<sup>2</sup><http://unity3d.com>, 27.11.2013

<sup>3</sup>[www.3dprofi.de](http://www.3dprofi.de)

built with fischertechnik instead of LEGO mindstorms<sup>4</sup>. Therefore, the chosen construction is Fischertechnik.

### Classification within the Engineering Model

The engineering model by Moore et al. describes four levels of which an engineering application can be classified. [44] The new mechatronic design environment is called **Fischertechnik Simulator** and can be used for developing mechatronic prototypes in the *composite component level* and the *modular machine level*. These levels are abstractions from the *device component level*. The **Fischertechnik Simulator** already implements devices from the *device component level* (e.g. static pieces, motor, button) and its to the user to design applications in the two overlying levels.



**Figure 3.4:** Engineering model by Moore et al. [44]

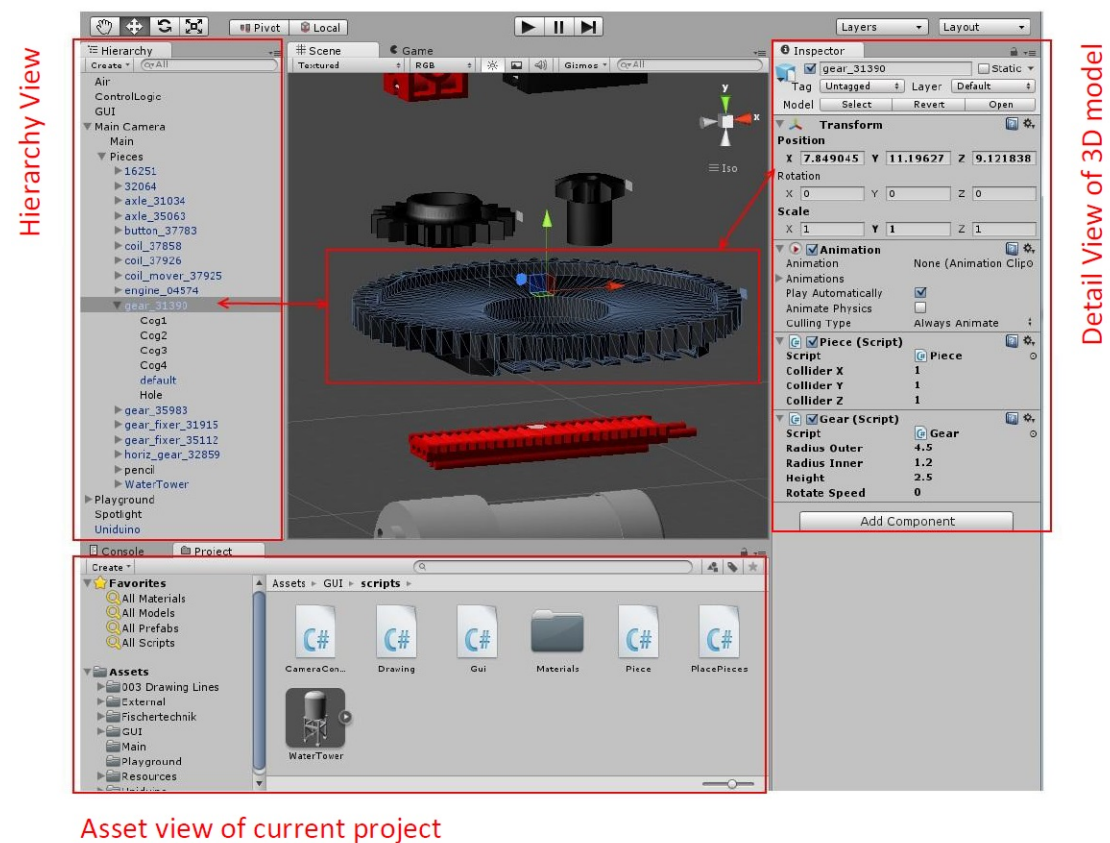
<sup>4</sup><http://www.kinder-technik.de/fischertechnik/eigene-modelle/modell-plotter.html>, 27.11.2013

## Principles of the Unity Game Engine

Unity is an engine for developing multiple platform games. But a software developed with Unity does not necessarily have to be a game. Unity provides rapid development of virtual applications in 3D. Applications can be exported to Windows PC, Apple iOS, MAC, Linux, XBox 360, Playstation 3 (PS3), Wii U and various browsers (requires a browser plugin). The free version is limited in its features (e.g. no mobile phone publishing, no real time shadows), but provides all development tools required for this implementation. Unity Technologies also operate an online *Asset Store* where developer can offer plugins and additional content for Unity (e.g. code libraries, textures, 3D models).

The next section is showing the principles of 3D model importing and coding behavior in Unity.

### Properties and Behavior of 3D Models



**Figure 3.5:** Unity development environment: The asset is the gear wheel, selected in the hierarchy view. It is depicted also in the 3D view and its details are listed in the detail view. Two classes (scripts), *Piece* and *Gear*, are attached to the gear wheel.

First, polygon models are not created in Unity. Therefore modeling software like Blender<sup>5</sup> or 3D Studio Max<sup>6</sup> are used. Common file formats for 3D models are wavefront (.obj)<sup>7</sup>, 3D Studio Max (.max, .3ds) or STereoLithography (.stl) [1]. These polygon models are loaded into the Asset view of a Unity project via Drag & Drop (see figure 3.5).

Assets (e.g. 3D model, sound, light) are moved into the game by dragging it from the asset view to the 3D view or the hierarchy view. In the hierarchy view, parent-child relations can be defined. The inspector view shows the properties of a loaded asset. The properties are: transformation information (position, rotation, scaling), surface properties (texture, color, shader), physical properties, animation settings and attached scripts. Attached scripts are describing the behavior of the asset with the source code (written in C#, JavaScript or Boo). To provide a main loop, a script (main.cs) can be attached to the camera or an empty object. Empty objects are placeholders for scripts. Attached classes which inherit from the Unity class *MonoBehaviour* are providing predefined methods like *OnMouseOver* or *OnCollision*. *OnMouseOver* is called when the user moves the mouse cursor over the 3D object, and *OnCollision* is called when another 3D object collides with the current 3D object. Important is the method *Start* which works as the constructor, and the method *Update* which is called every drawing frame. These methods are override-able.

## Specification of the Simulation Environment

The simulation environment is the virtual “playground“, where mechatronic Fischertechnik applications can be built and simulated. The components of the simulation environment (menu, camera movement, build mode, control mode, new, load, save) are specified with sketched figures, use cases and user stories.

### The Menu

Figure 3.6 sketches the start screen of the Fischertechnik Simulator. On the right in a vertical slider bar, the Fischertechnik construction pieces are placed for selection (only visible in build mode). The base plate in the 3D view is empty and provides slots for plugging pieces with knobs on it. The underlying features of the menu options *New*, *Load*, *Save*, *Build*, *Control*, *Arduino* and *textitRun* are described in the next sections. The menu option *Filename* is a placeholder for showing the name of the current simulation project.

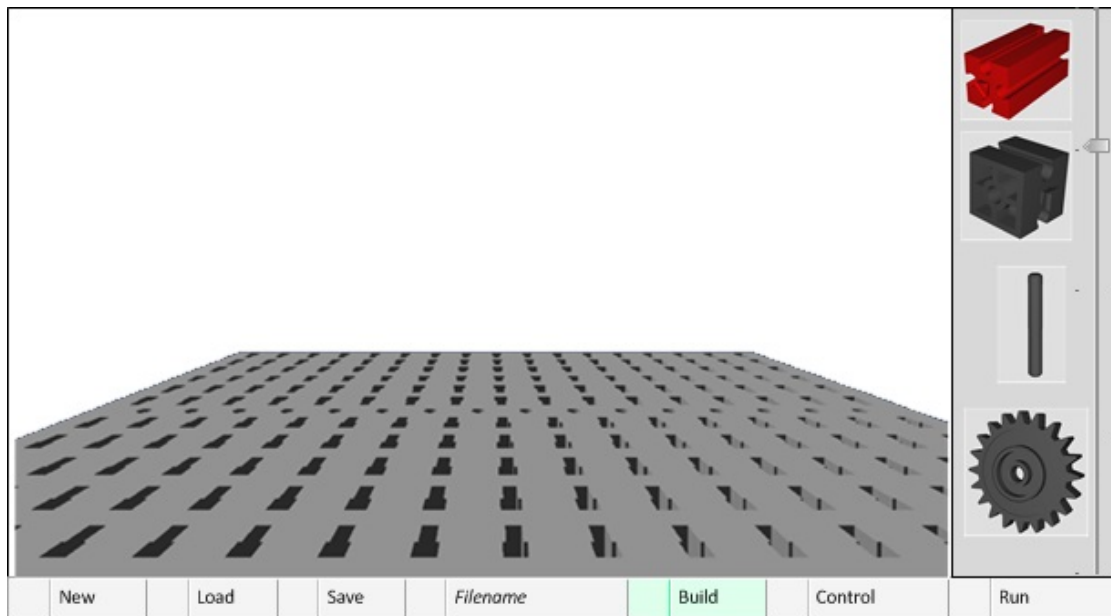
---

<sup>5</sup>[www.blender.org](http://www.blender.org)

<sup>6</sup>[www.autodesk.de](http://www.autodesk.de)

<sup>7</sup><http://people.sc.fsu.edu/~jburkardt/data/mtl/mtl.html>, 27.11.2013





**Figure 3.6:** Menu of the Simulation: Building Mode

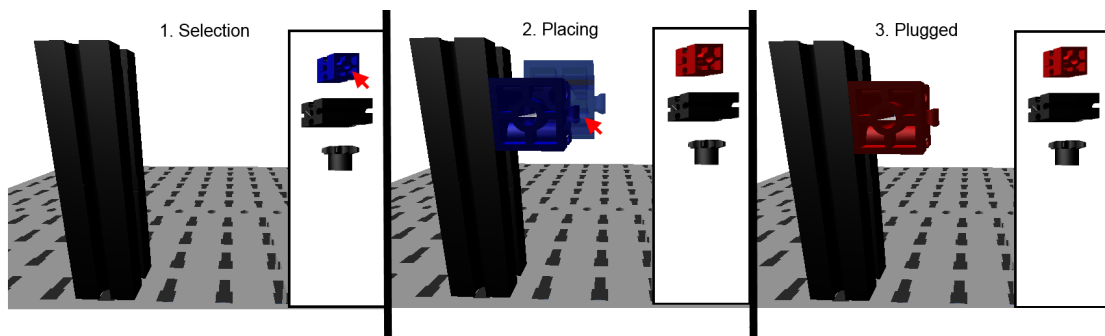
### Navigation

The navigation of the 3D screen is inspired from the Unity development environment. The user can adjust the view by:

- Scrolling the mouse wheel: Zoom in/out.
- Pushing the mouse wheel while moving the mouse: Strafe left/right or up/down.
- Holding right mouse button while moving the mouse: turn left/right or up/down.

### Build: Placing Fischertechnik Pieces

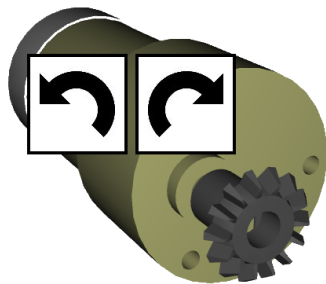
The pieces follow the connection principles depicted in figure 2.2. Pieces from the selection box are placed on the base plate or on other pieces. Placing a piece consists of the following steps: First, the user selects a piece from the slider by clicking on it. Second, a copy of this piece is generated. Third, the copy can be navigated with the mouse and scroll wheel over the 3D screen. Fourth, a left click places it on the current position or a right click deletes the copy (figure 3.7).



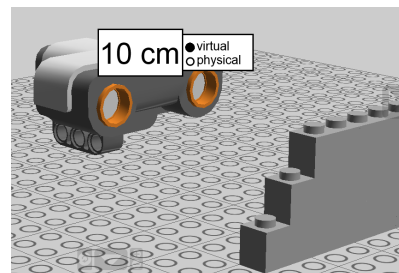
**Figure 3.7:** Use Case: Placing a static object in the 3D view of the simulation environment.

### Control: Configuring Actuators and Sensors

Actuators and sensors can be controlled directly in the building menu, and can also be configured for orchestration behavior in the control mode. The direct control is for an explorative trial and error development of the prototype. This way the user can simply turn actuators on and off, or also can see immediately the status of the sensors (figure 3.8 and 3.9)

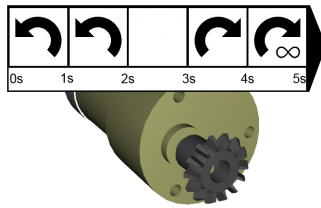


**Figure 3.8:** Explorative actuator control on the example of a motor: When moving the mouse over the motor, buttons for rotations pop up in front of the motor. The user can then rotate the gear on the motor by clicking these buttons.

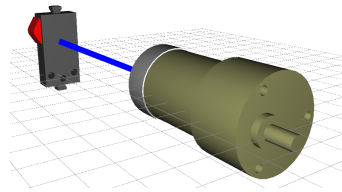


**Figure 3.9:** Explorative sensor read on the example of a distance sensor: A textbox which is visible in the 3D view shows the measured distance. The user can also choose if the distance is measured in the virtual environment, or originates from a physical sensor.

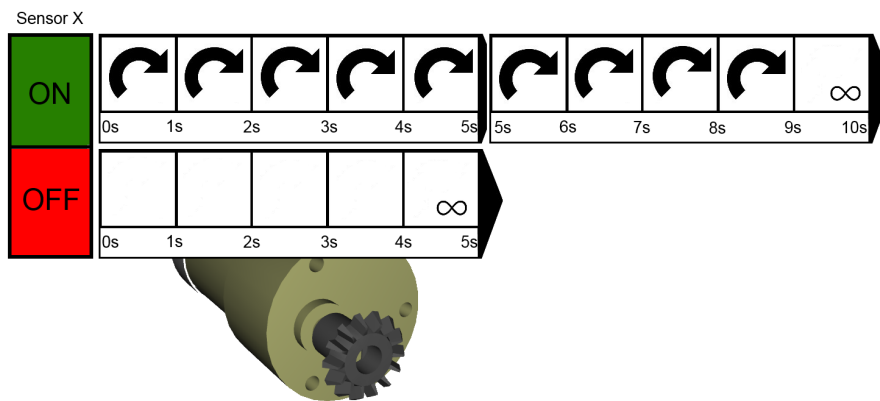
For scheduling and orchestration in the 3D view, the actuators of a prototype are visually enriched by interactive timing diagrams. In these timing diagrams, the behavior of the actuator can be defined over a timespan (figure 3.10). If a sensor is attached to an actuator, a control behavior for each possible status of the sensor can be set (figure 3.12). In the GUI menu, the defined schedule plan can be started, paused and reset.



**Figure 3.10:** Planing actuator status over the time on the example of a motor: A time frame covers 5 seconds of motor behavior. With the indicated arrow right of the time frame, further time frames can be added. By clicking a status box within a time frame, the status of rotation can be defined for a specific second. In the above example, the motor turns counter clockwise in the first two seconds, then stops for one second and then turns clockwise forever. The most right status box has always the indication of endless continuation.



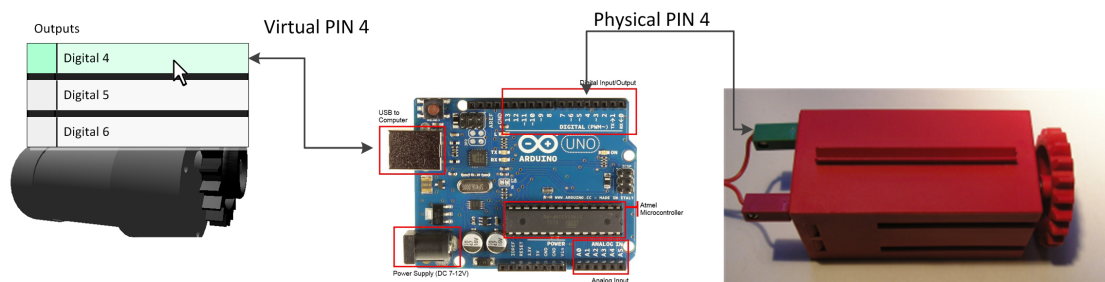
**Figure 3.11:** Setting an actuator in dependency of a sensor: Here, a digital sensor (button) is attached to the motor by drawing a line between the button and the motor in the 3D view. The GUI has an option to show or hide the sensor/actuator connections.



**Figure 3.12:** Actuator status depending on the sensor: In the 3D view, the user can attach a sensor to an actuator. In the picture the behavior is similar to the hand dryer depicted in figure 2.3 from section Prototyping Environments2.3. If the sensor is triggered to ON, the motor runs clockwise for 9 seconds. On sensor status OFF, the motor does not rotate.

## Synchronizing virtual and physical devices

To achieve a synchronous run of the virtual and the physical environment, the devices have to be assigned to each other. After a connection to the Arduino board is established, the user can assign the virtual devices to the physical pins of the Arduino board in a drop down menu. Following this choice the user then has to connect the physical device to the corresponding hardware pin on the Arduino board.



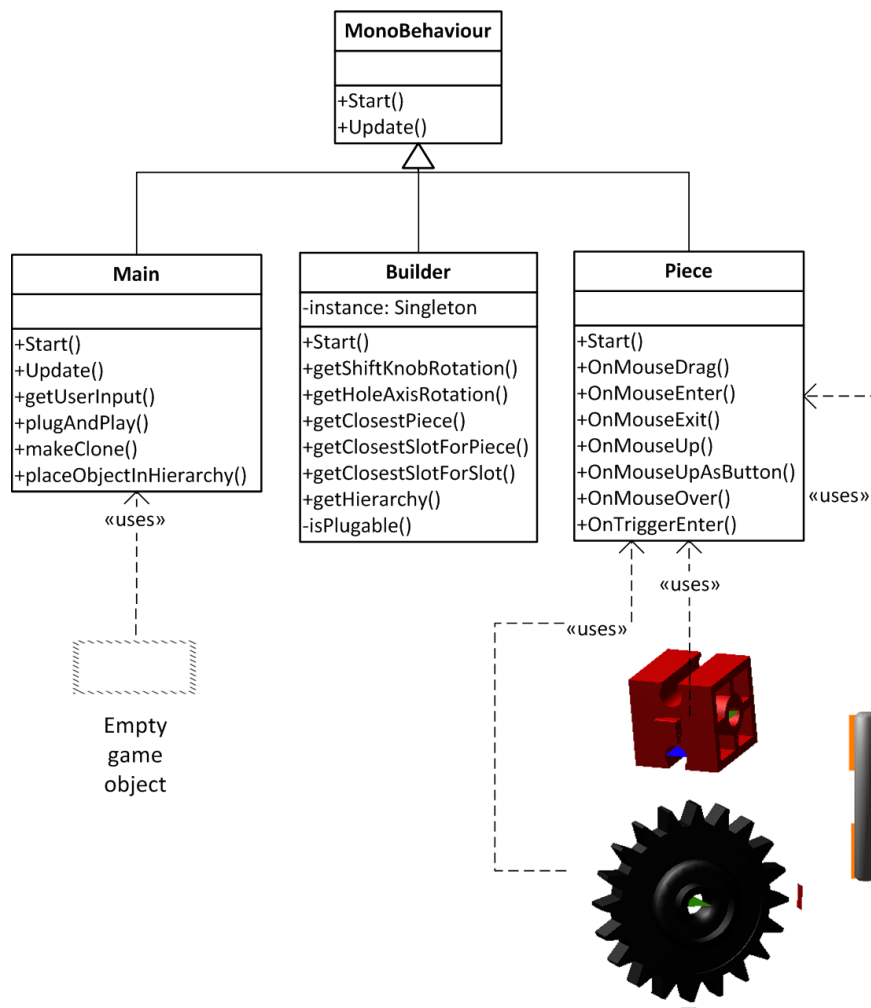
**Figure 3.13:** Selecting a pin for the virtual motor will configure the corresponding pin on the hardware side as output pin. It remains to the user to connect a physical motor to this pin on the Arduino board.

## Implementation of the Simulation Environment

This section gives a software design for an implementation of the simulation environment. Each important feature refers to a use case from section 3.2 and is described separately in a subsection. For each feature, first a class diagram is given, and second a sequence diagram explains how the use case can be implemented. The given class diagrams do not show all its methods. Only methods required for the use case are given. The following sequence diagram is depicting the flow of the use case implemented by object oriented classes and methods.

### Placing 3D Objects

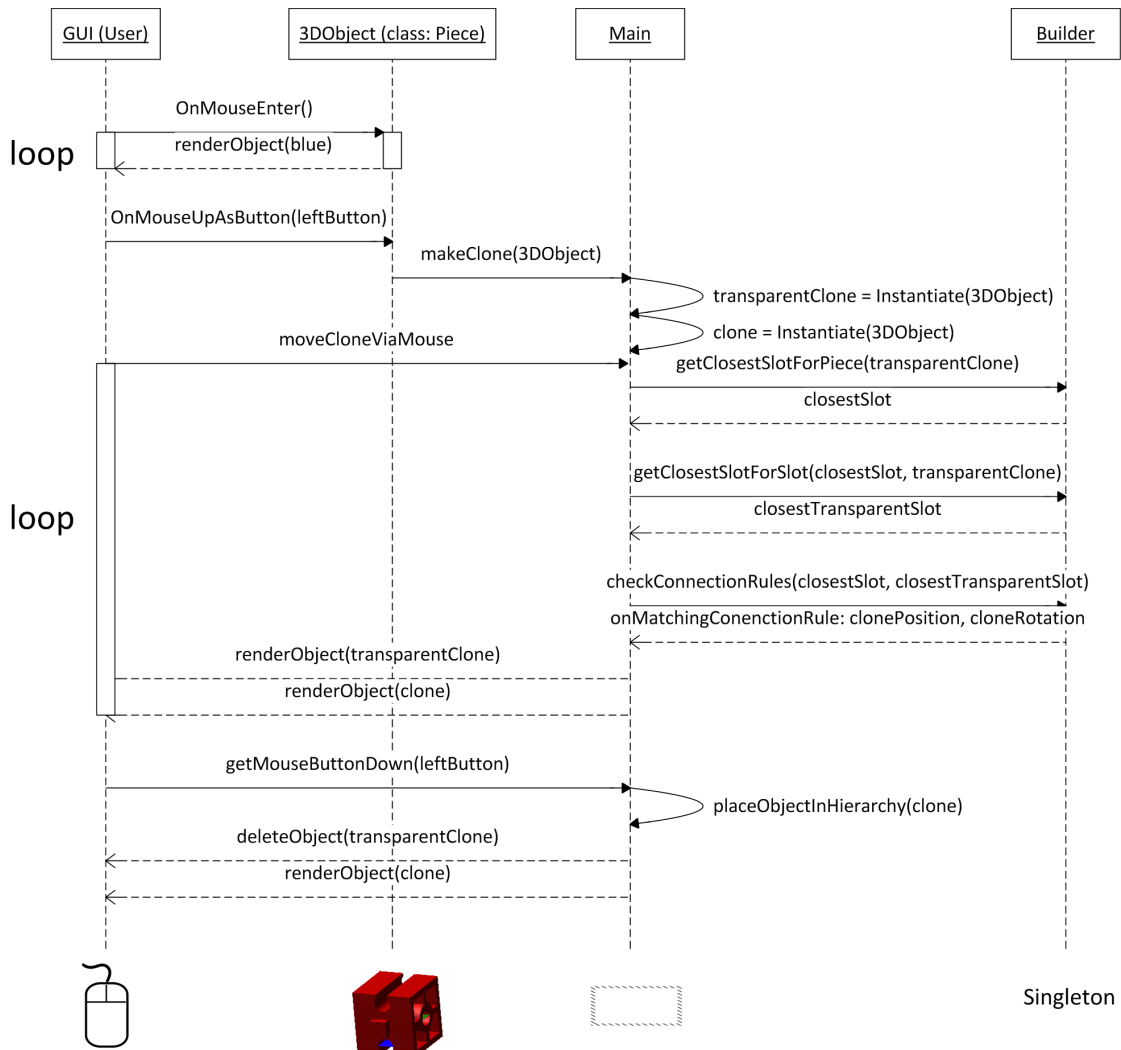
**The class diagram in figure 3.14:** Each 3D model implements (uses) an object of the class *Piece*. The methods of *Piece* provide input operations (via mouse) on the 3D models. *Main*, *Builder* and *Piece* inherit from the Unity class *MonoBehaviour*. Methods which first letter is capitalized are overridable methods from *MonoBehaviour*. *Main* is also implemented by a object in the 3D space. But this object is an empty game object, which means that it has no polygon model and is not visible to the user. Since *Main* serves as main-loop for the simulation, the loop cycle can only be run by the method *Update* if *Main* is implemented by a game object. *Builder* is a singleton class that provides helper methods for *Main*.



**Figure 3.14:** Class diagram for building a static prototype

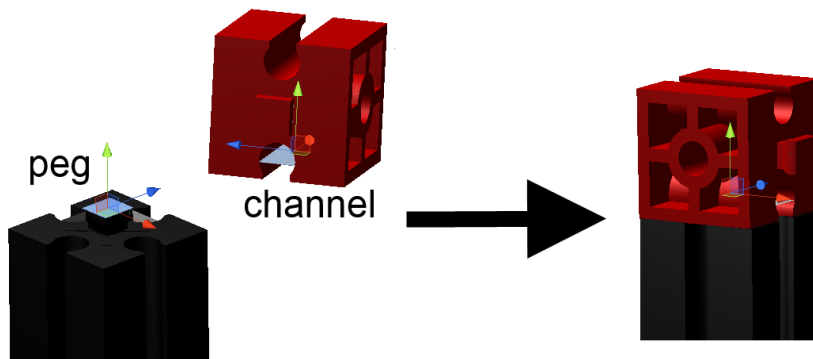
**The sequence diagram in figure 3.15:** The *GUI* represents the 3D screen where the user navigates with the mouse. First, a 3D object is selected from the selection menu (*OnMouseEnter* and *OnMouseUpAsButton*). *Main* is always waiting for appearing objects in the 3D view. The class *Piece* (attached to the 3D object) calls a *Main* method creating two clones of the 3D object. The *transparentClone* is for visualizing the GUI navigation of placing the new item. In a loop, *Main* continuously calculates the closest correct connection of the *transparentClone* to an yet already placed 3D object. The object *clone* is positioned at the closest possible calculated connection. (A working plate with connection slots for initial connections is always placed in the 3D view). The closest possible connection is determined in a loop which contains three steps: First, it is calculated which slot from an already placed 3D object is closest to the *transparentClone*. In the second step the closest slot from the *transparentClone* to the previously found *closest slot* is calculated. The third step determines if the calculated two slots are following a fischeretchnik

connection principle. If yes, the *clone* is rendered at the matching position. A final mouse click by the user places the *clone* at the current position and deletes the *transparentClone*



**Figure 3.15:** Sequence diagram for building a static prototype

**Connection with slots in detail:** A slot is the abstract description of all types of connection interfaces (e.g. pegs, holes, channel, axle described in figure 2.2). In the 3D environment a slot is generally a two dimensional plane. This planes are enriching the 3D objects and defining therefore the possible connections to other 3D objects. The name of a 2D plane is used for exact identification of the slot type.



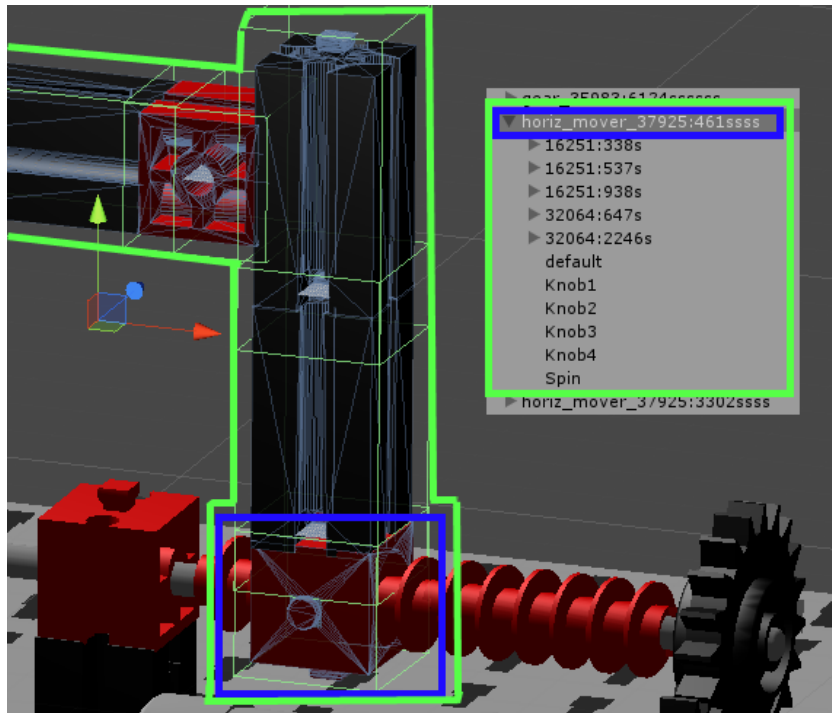
**Figure 3.16:** Slot connection detail.

For placing the red 3D object on the black one, the red 3D object has to be rotated and translated till the axis orientation and position of the channel and the peg are the same.

$$objectRed.angle = objectBlack.slots.peg.angle - (objectRed.slots.channel.angle - objectRed.angle) \quad (3.1)$$

$$objectRed.pos = objectBlack.slots.peg.pos - (objectRed.slots.channel.pos - objectRed.pos) \quad (3.2)$$

**Hierarchical placing in detail:** When a 3D object is placed by releasing the mouse button, it has to be attached to the correct position in the hierarchy model of Unity. The method *placeObjectInHierarchy* knows that for translating 3D objects, new 3D objects have to be attached as a child to the translating object. This is required because a motion of the translating object results in the same motion of the child objects. (see figure 3.17)

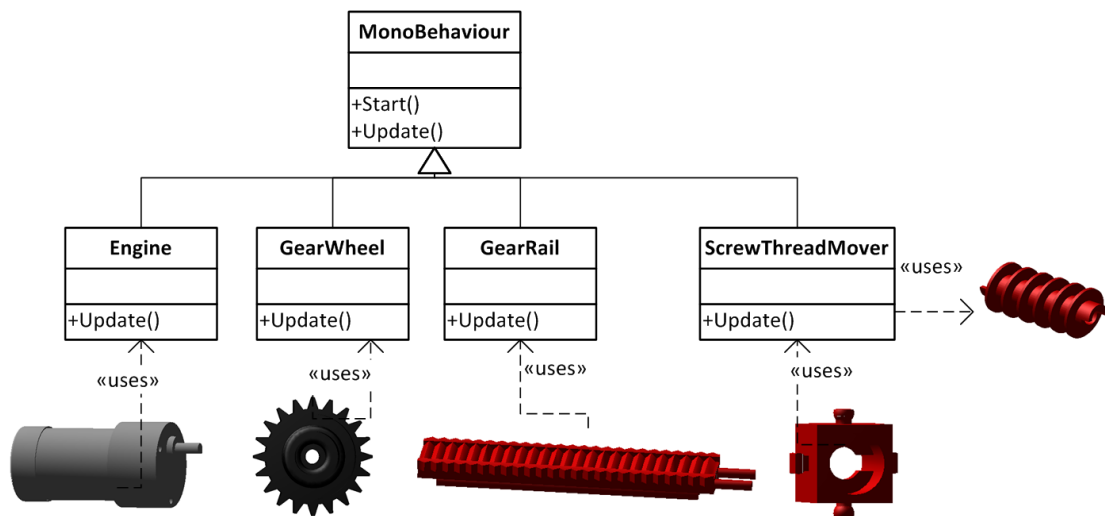


**Figure 3.17:** Hierarchy model detail: The 3D object *horizontal mover* (framed blue) is an object that translates on gear rotation. So all the 3D objects connected to *horizontal mover* are also translating because they are attached as child objects to the *horizontal mover*.

### Calculate Motion

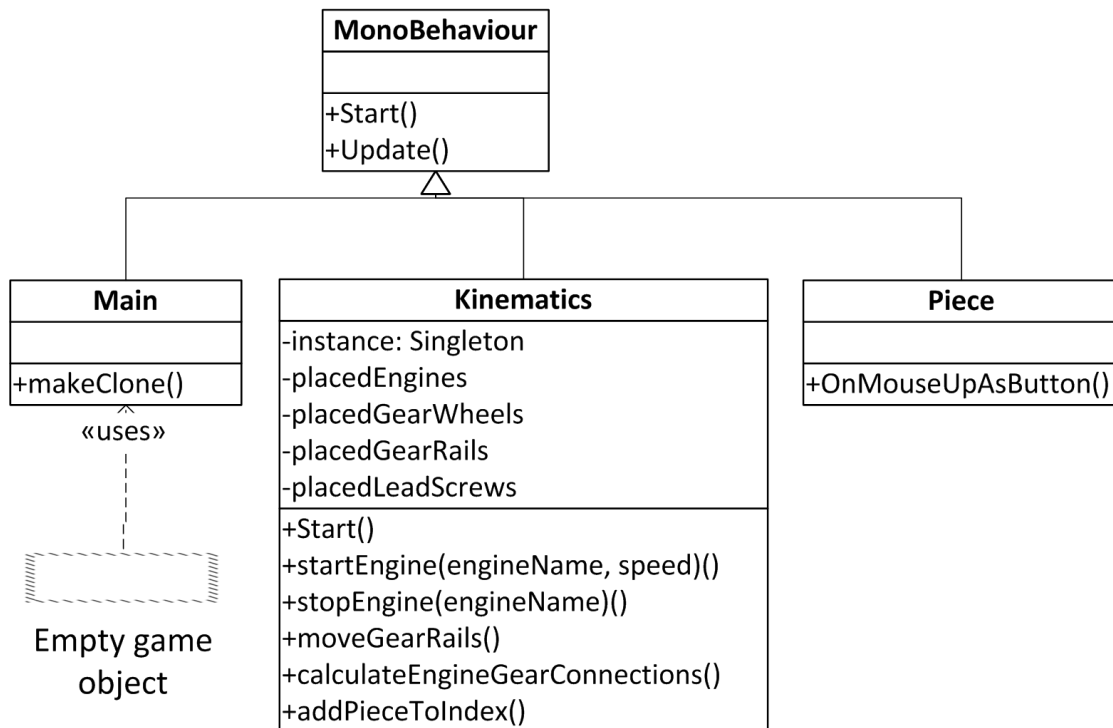
**Class Diagram of 3D motion objects:** Different types of 3D objects are carrying out motion operations. The source of motion is always an engine with an attached gear wheel which results in rotational motion. Translations of fischertechnik 3D objects can be achieved with use of gear rails or screw threads. Both transfer rotation from gear wheels to translation. Figure 3.18 depicts the classes attached to the 3D objects.





**Figure 3.18:** 3D objects which handle motion: The class *Engine* is responsible for turning the engine and its further attached gear wheel ON and OFF. *GearWheel* operates rotations in the method *Update* and *GearRail*. *Update()* handles translations. The class *ScrewThreadMover* also calculates translation on the 3D object screw thread.

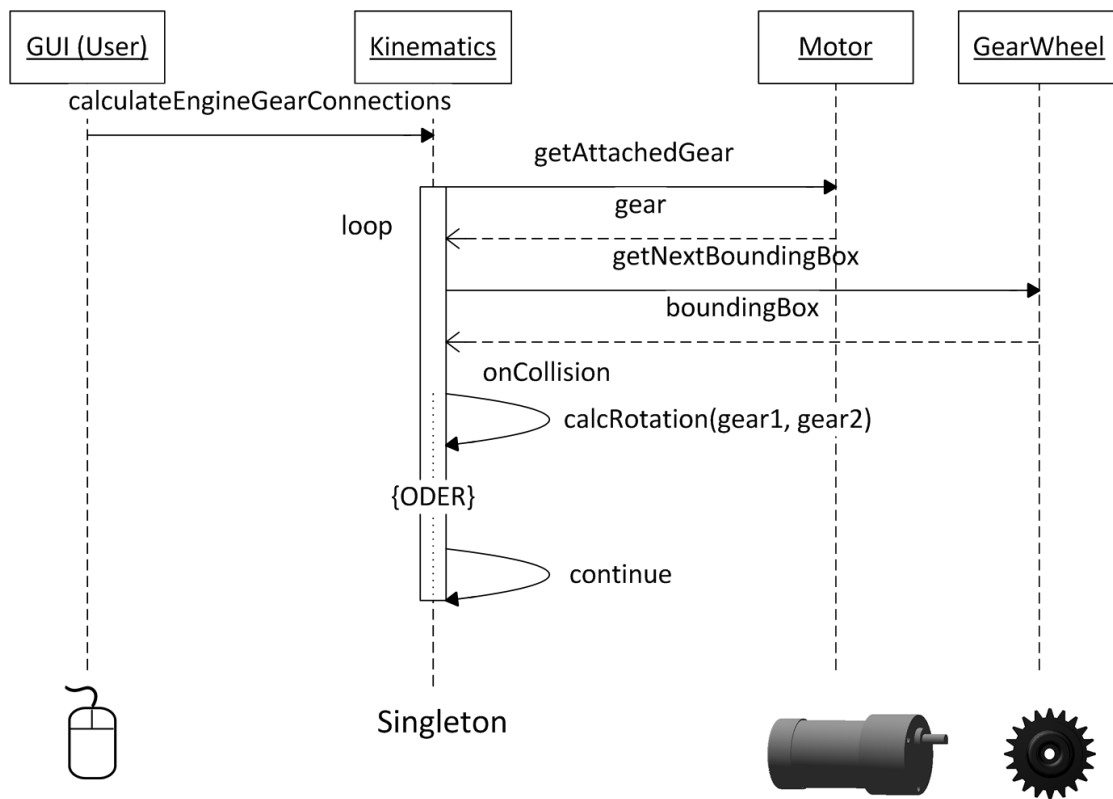
**Class Diagram of Motion Calculation:** The singleton *Kinematics* with its attributes and methods is responsible for motion calculations. In the attributes the already placed 3D motion objects are stored. They work as a lookup repository for new 3D motion objects that need to know if they are inflicted by motion of these 3D objects.



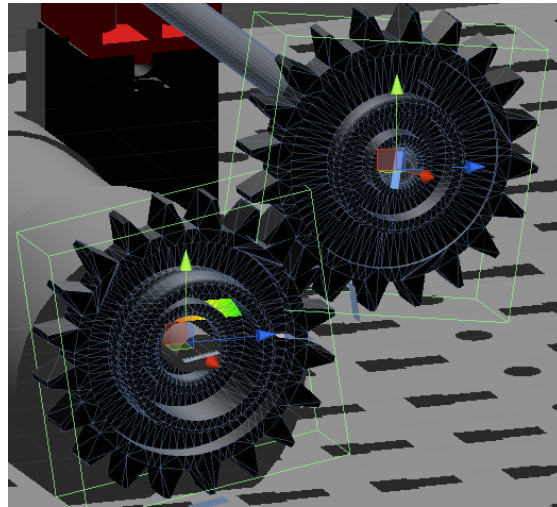
**Figure 3.19:** Essential classes and methods for motion calculation

**Sequence Diagram of Motion Calculation:** The method *calculateEngineGearConnections* calculates and sets all motion parameters. The call of this method is required before a motor is switched ON. The class *Kinematics* then calls for each gear wheel attached to a motor the following procedure:

Every other gear wheel in the 3D scene is checked if it collides with the gear wheel on the motor. On collision, the rotation speed of the colliding gear wheel is calculated based on the gear wheels diameter and the rotation speed of the motor gear wheel. This new found gear wheel works as a starting point for searching other colliding gear wheels. This algorithm is depicted in the sequence diagram 3.20. Figure 3.21 displays two gear wheels with their bounding boxes and axis orientation.



**Figure 3.20:** Sequence diagram for gear wheel rotation calculation.



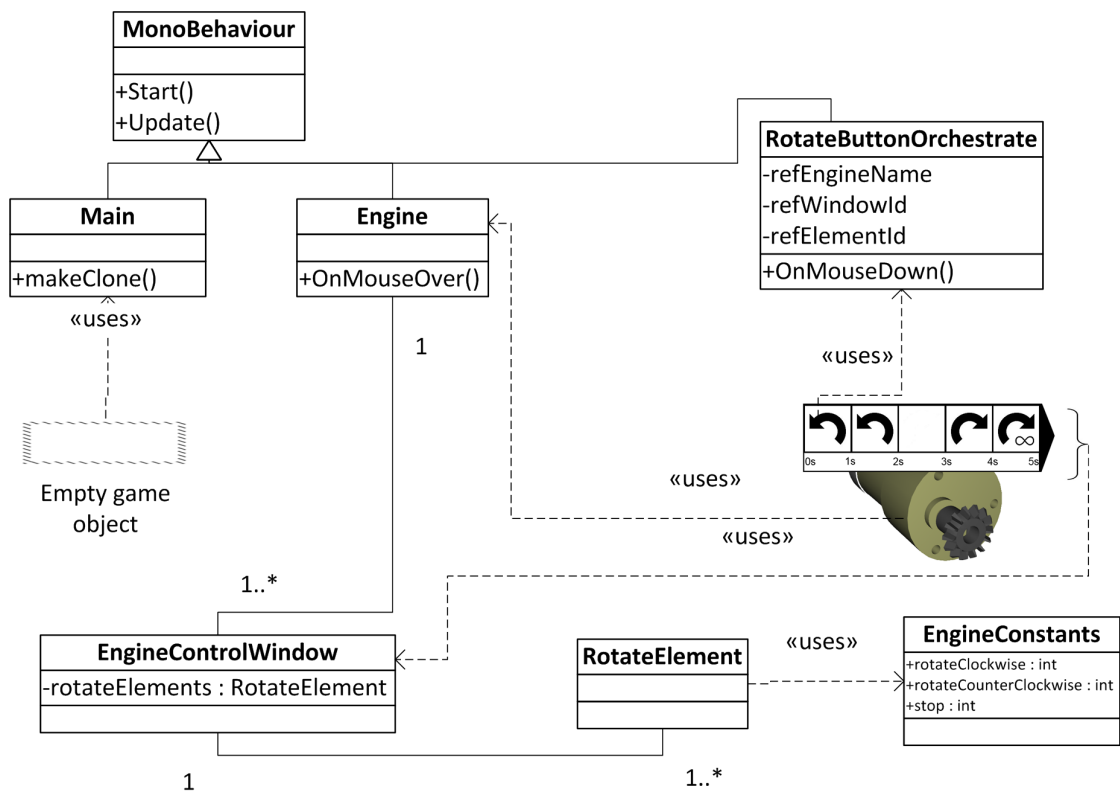
**Figure 3.21:** Two gear wheels with bounding box and axis orientation: The two light green boxes are the bounding boxes of the gear wheels. In this setting they are colliding and are therefore taken into the rotation calculation. Both rotate around their local x-axis (red vector). By construction both x-axis vectors are parallel, but the angle between these two vectors can be 0 degrees (depicted in this figure) or 180 degrees. On 0 degrees the rotation speed has to be multiplied by -1 to let the gear wheel rotate in the other direction. On 180 degrees the correct rotation direction results immediately.

Calculating the motion of translating pieces (gear rail, screw thread) follows the same principle. But the difference lies in the interval of recalculation. For translating pieces, it has to be checked periodically (in every frame) if the piece with translation motion still collides with the gear wheel or the screw thread mover.

### Orchestrate a Simulation

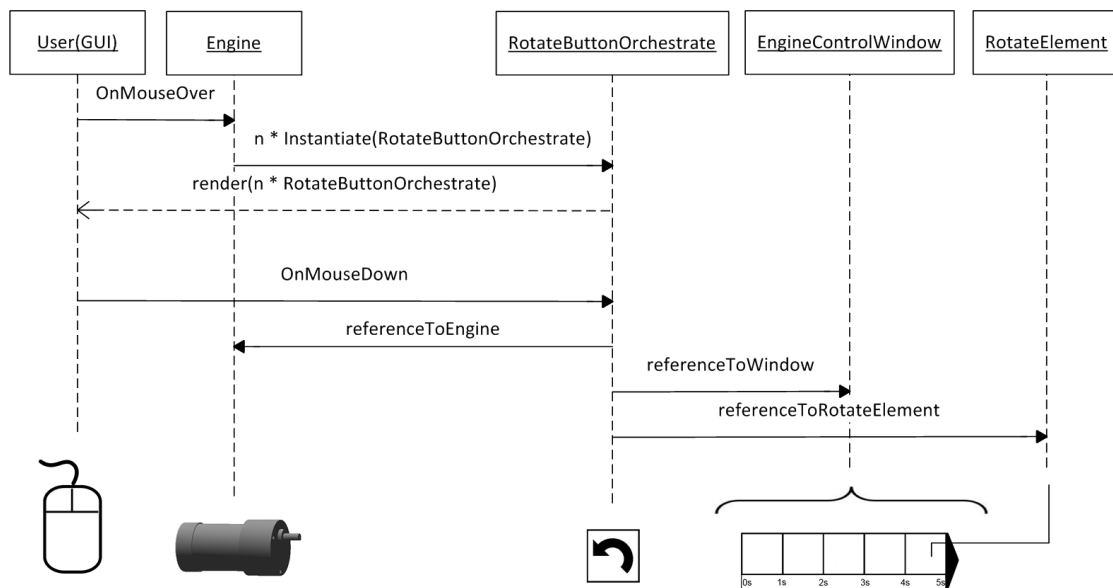
A simulation run can either be designed for fully automatic runs like the control of a plotter, or can be designed in a reactive way which involves sensors (e.g. fan control from section 2.3).

**Class Diagram for defining an automatic simulation run:** When moving the mouse over a motor, the simulation instrument for orchestrating control flows is popping up. This simulation instrument consists of *EngineControlWindow*'s. One *EngineControlWindow* holds five *RotateElement*'s. A *RotateElement* defines the motor status for the duration of one second. In the 3D screen an *EngineControlWindow* has on the right an arrow button to generate a further control window. The class *RotateButtonOrchestrate* is attached to the visualization for a one-second motor control (rotating arrow) and holds a reference to the motor, a control window, and a rotate element.



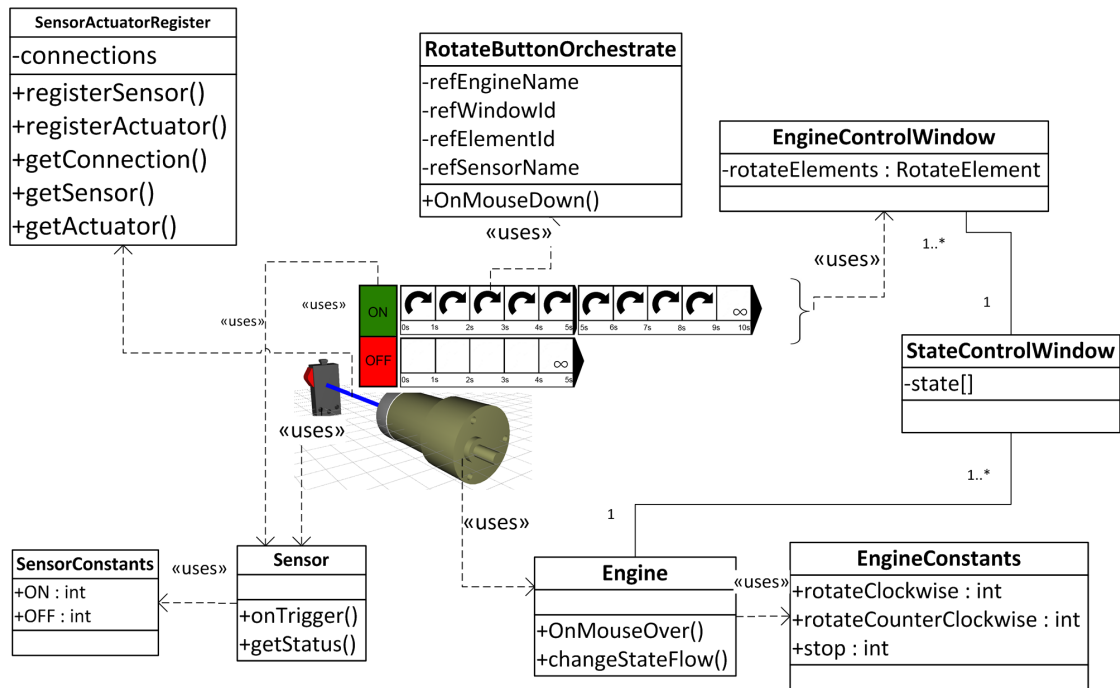
**Figure 3.22:**

**Sequence Diagram for defining a automatic simulation run:** The method *OnMouseOver* is called when the user enters with the mouse the bounding box of the motor. Clickable rotate buttons (*RotateButtonOrchestrate*) are rendered over the motor. Initially five rotate buttons which are assigned to one *EngineControlWindow* are rendered. If the user has previously extended more engine control windows by clicking on the arrow on the right, the number of rotate buttons is extended by the number of engine control windows multiplied by five. Clicking on a *RotateButtonOrchestrate* changes the state to *rotateClockwise*, *rotateCounterClockwise* or *stop*. Then the reference is set to the motor, engine control window and the position in the engine control window (*RotateElement*). When the final simulation run is triggered by the user, the defined rotation behavior in the engine control windows is executed for each motor.



**Figure 3.23:** Sequence diagram for a planned simulation run.

**Class Diagram for adding sensor behavior:** An engine behavior depending on a sensor is displayed visually via a direct connection line between the sensor and the actuator in the 3D screen. Depending on the sensor, different states are available. Here, the digital sensor implements the states ON and OFF. The *SensorActuatorRegister* stores the connections between sensors and actuators. The *Engine* has a new data structure called *StateControlWindow* for storing control flows for each possible sensor state. The *Sensor* can periodically (in each frame) check by *SensorActuatorRegister.getSensor()* if the sensor state has been changed. On change, the engine switches the control flow by changing the *StateControlWindow*.



**Figure 3.24:**

**Sequence Diagram for registering a sensor:** The *SensorActuatorRegister* stores a connection between a sensor and an actuator. This happens by dragging a line from a sensor to an actuator. *OnMouseDown* finally registers the connection.

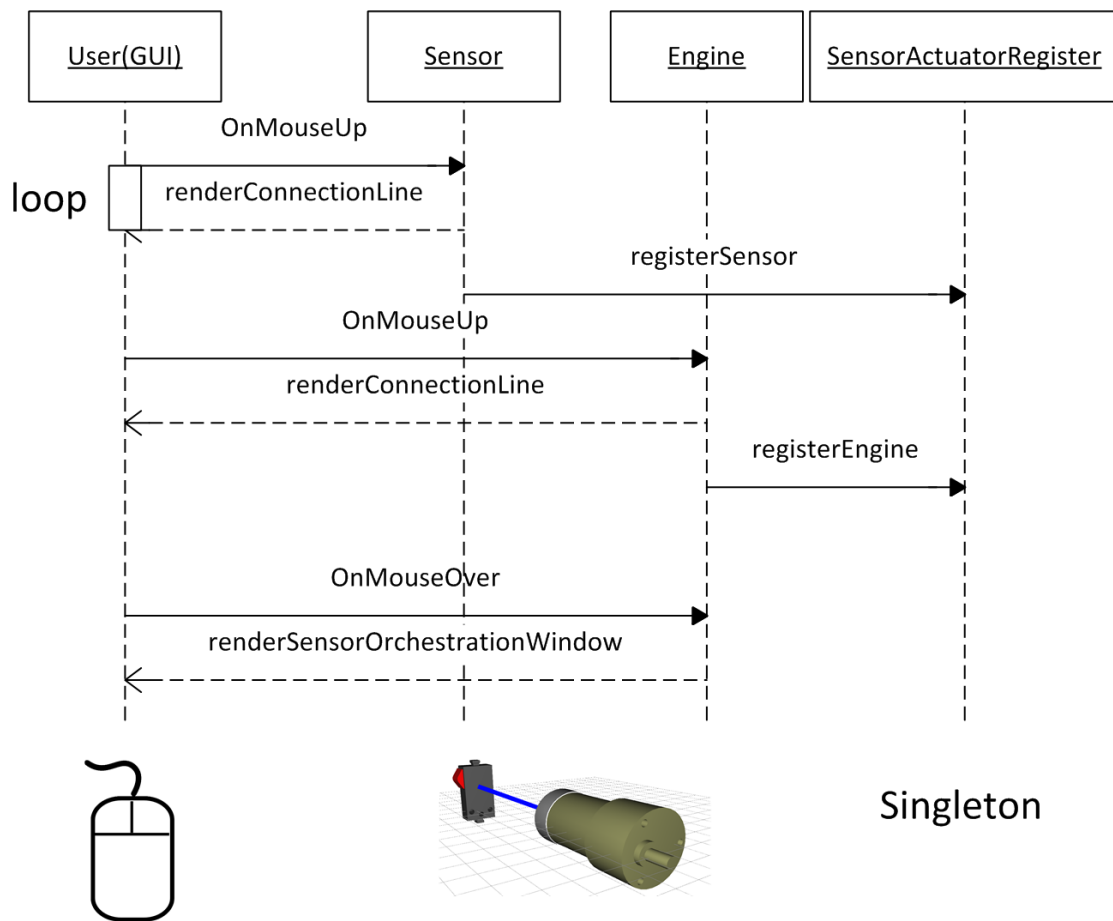


Figure 3.25:

### Specification of Fischertechnik Synchronization

The goal is to transmit the motion and control information of the virtual prototype to the real prototype. The configuration which pin of the arduino board corresponds to a certain virtual representation of an actuator or a sensor has to be set via a drop down menus.

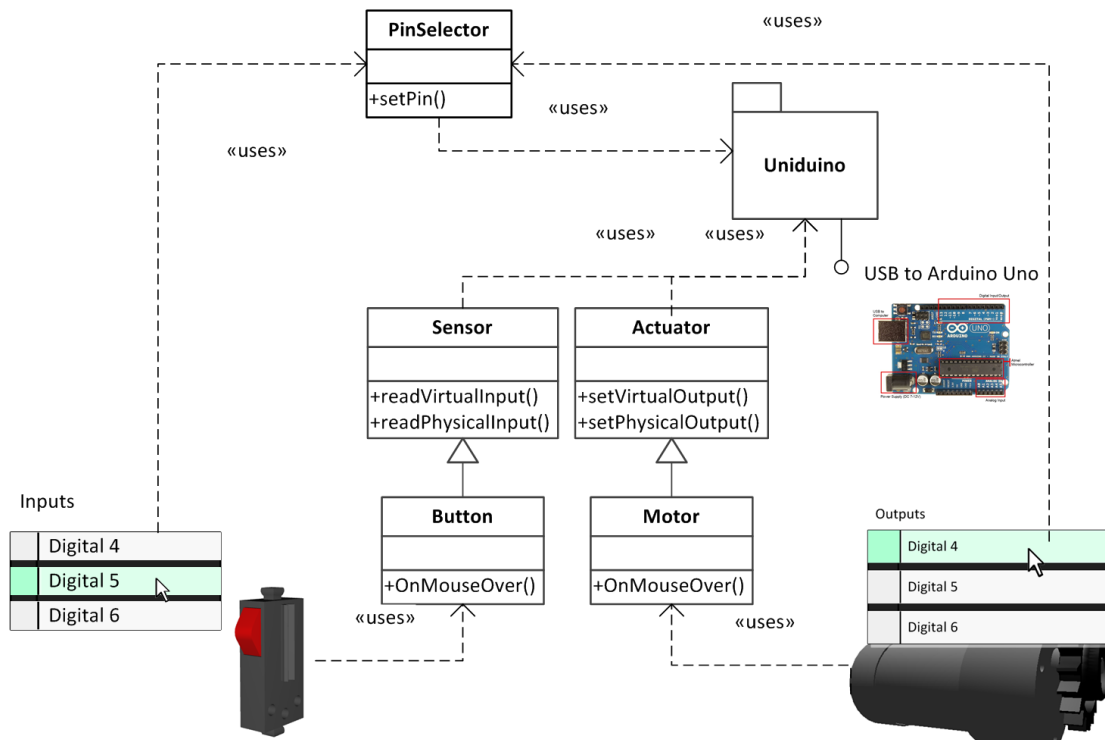
### Implementation of Fischertechnik Synchronization

**Class Diagram for synchronization:** To connect the virtual sensors and actuators to its physical representations via the Arduino board, the Unity asset *Uniduino*<sup>8</sup> is used. It lets the virtual environment connect to the arduino board via USB. The GUI features a pin assignment mode where visual pin selection menus pop up on `OnMouseOver()` on a sensor or an actuator. The method `readVirtualInput` calls sensor data directly from the state of the virtual sensor. On the

<sup>8</sup>[www.uniduino.com](http://www.uniduino.com)

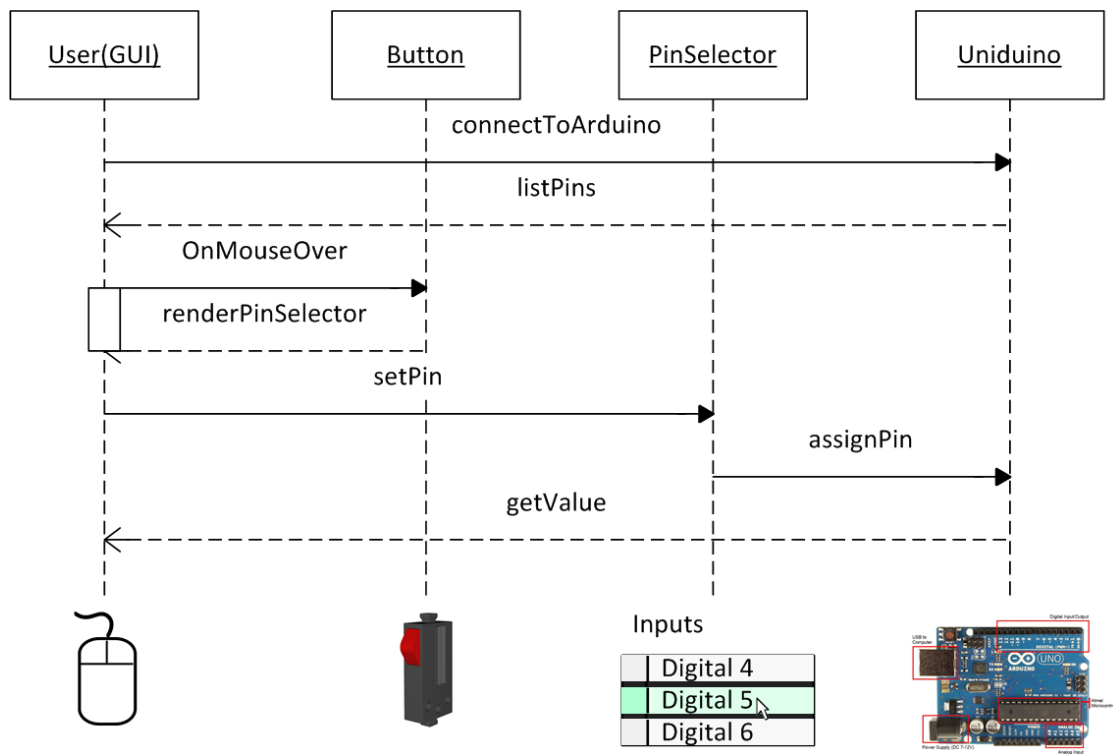


other hand the method *readPhysicalData* reads the sensor data from the physical device via the *Uniduino* package.



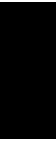
**Figure 3.26:** Class diagram for synchronization.

**Sequence Diagram for synchronization:** Before using the pin assignment, the user has to establish a connection to the arduino board via the *Uniduino* package. A successful connection returns the available pins for reading and sending data. *OnMouseOver* on a sensor renders the pin selector menu. Selecting a pin assigns the pin to the chosen sensor. Physical sensor data can then be read via the *Uniduino* package and be used for further progress.



**Figure 3.27:** Sequence diagram for synchronization.

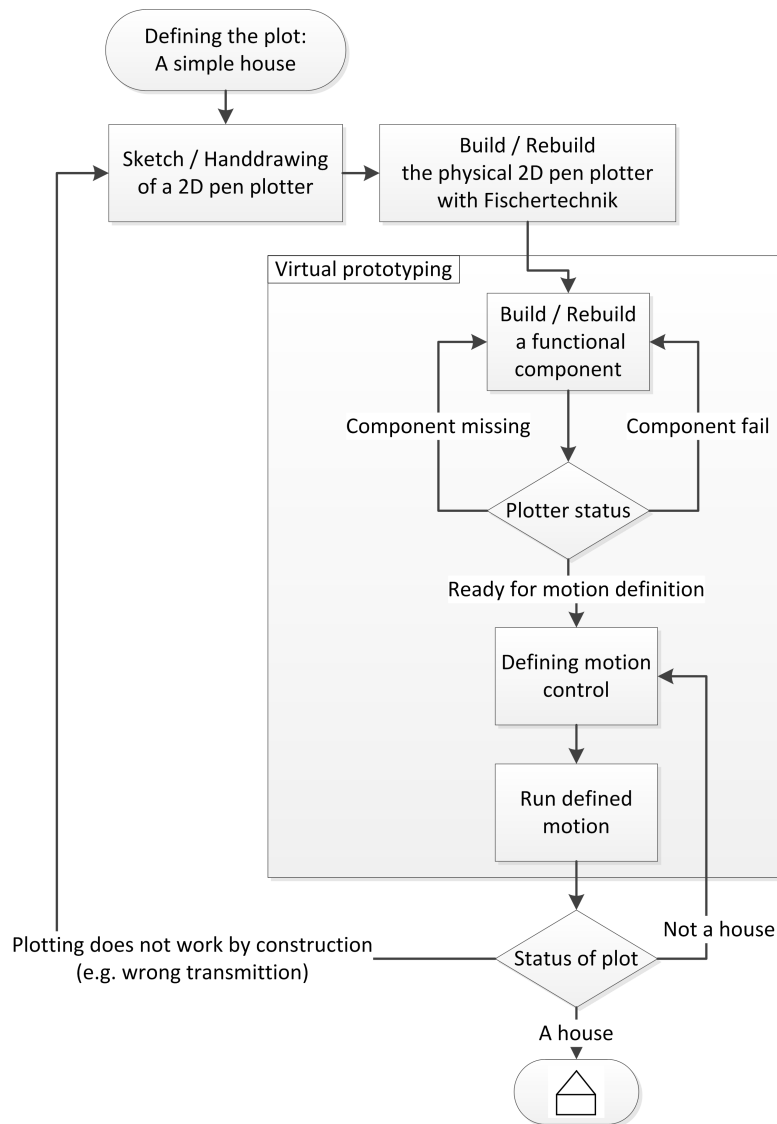
CHAPTER 4



# Evaluation and Critical Reflection

## 4.1 Evaluation on the Example of a 2D Plotter

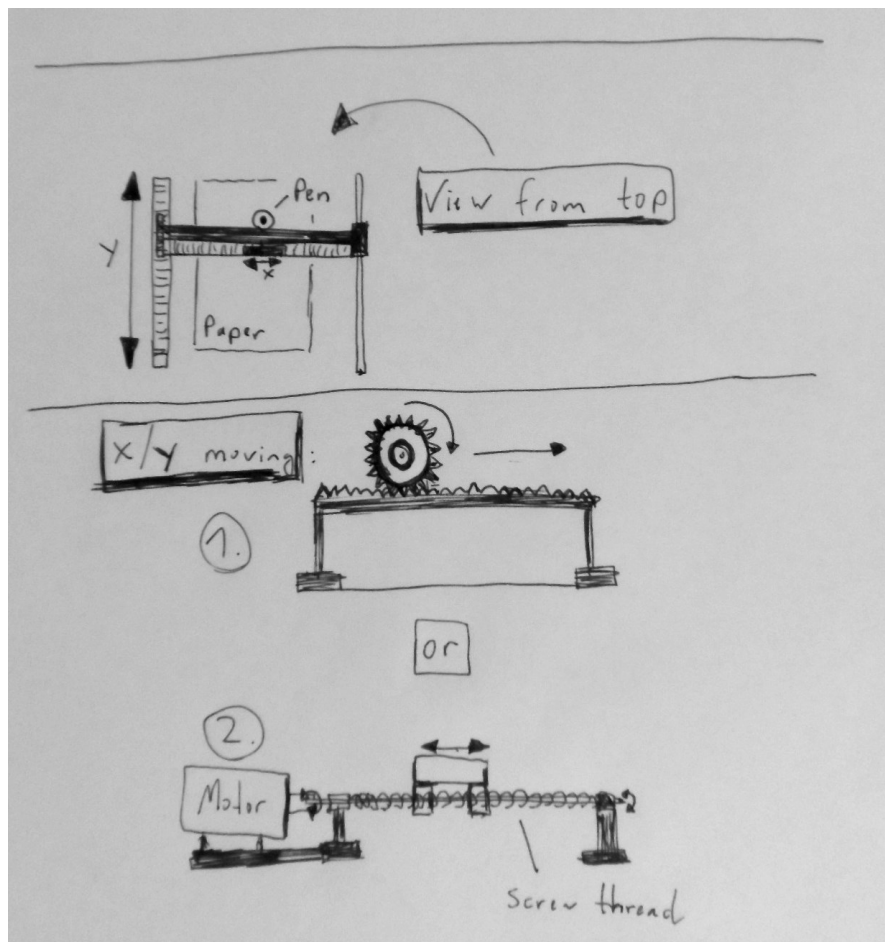
### Overview of the Prototyping Process



**Figure 4.1:** The prototyping process of designing the 2D plotter which plots a house in the virtual and physical environment.

Figure 4.1 depicts the whole prototyping process with its iterations and steps. After sketching the plotter and the desired plotting object (a house) a physical version of the plotter is build. Then the virtual prototyping starts. Each functional component is tested in the virtual environment after building (e.g testing the transmission of the gear rail via motor rotation). When the virtual plotter is designed similar to the sketch a motion definition for plotting a house is designed and tested iterative. In case that no motion definition can achieve the correct plot, the 2D plotter has to be redesigned.

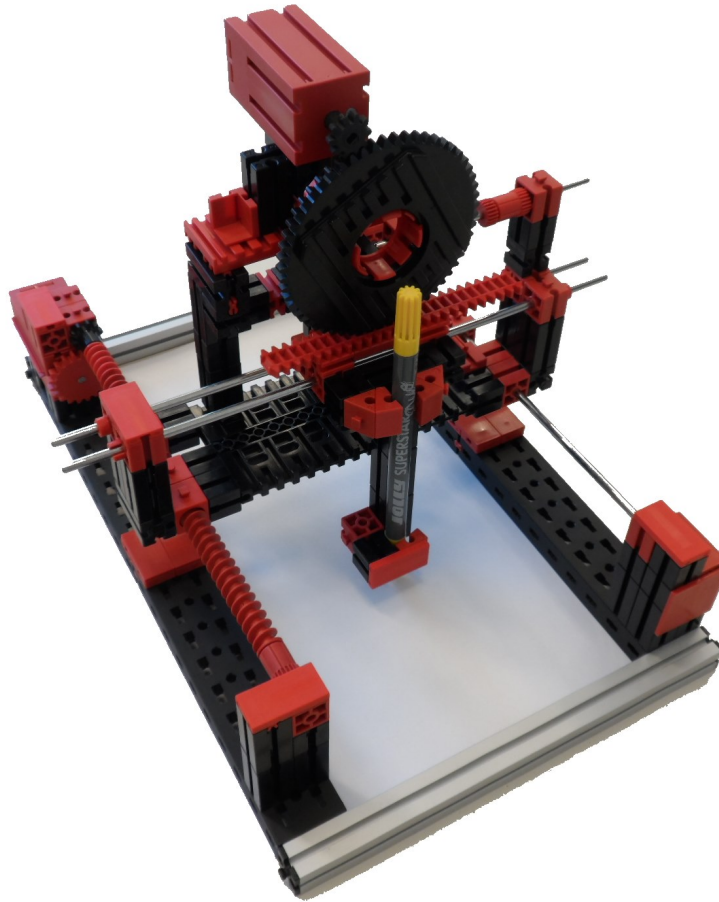
## Preparation



**Figure 4.2:** A hand drawing sketch of a 2D pen plotter including thoughts about the axis movement.

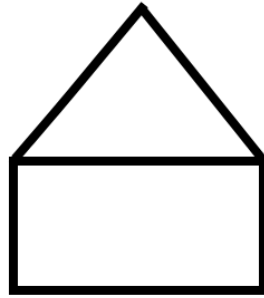
In figure 4.2 a top view of the 2D pen plotter with the axis  $x$  and  $y$  is sketched. For movement along the  $x$  and  $y$  axis two possibilities are suggested. First, a gear wheel rotates along a gear rail which is fixed on the ground. In this case the motor would be rotating the gear real directly

or via a transmission. Or second, a screw thread (rotated by a motor) which moves a nut along the axis. In this case the motor would be fixed to the ground.



**Figure 4.3:** A physical 2D plotter built with Fischertechnik.

To validate the implementation of the mechatronic design environment, a physical 2D plotter built with Fischertechnik serves as reference. Both versions for movement along the axis (see figure 4.2) are built into this first prototypical reference. The goal is to build this 2D plotter in the virtual environment, define a motion control and execute the motion (resulting in a plot) on the virtual and the physical 2D plotter. The motion control should be configured to draw a two dimensional house (see figure 4.4). The 2D plotter consists of static elements, actuators, hierarchical motion representation, different gear wheel transmissions, horizontal gears and screw threads.



**Figure 4.4:** The goal for the virtual and physical 2D plotter: Plotting a two dimensional schematic of a house. It requires motion of only one motor for the straight lines and motion of two motors (results in hierarchical representation) for the diagonal rooftop lines. The motors have to be able to rotate clockwise and counterclockwise.

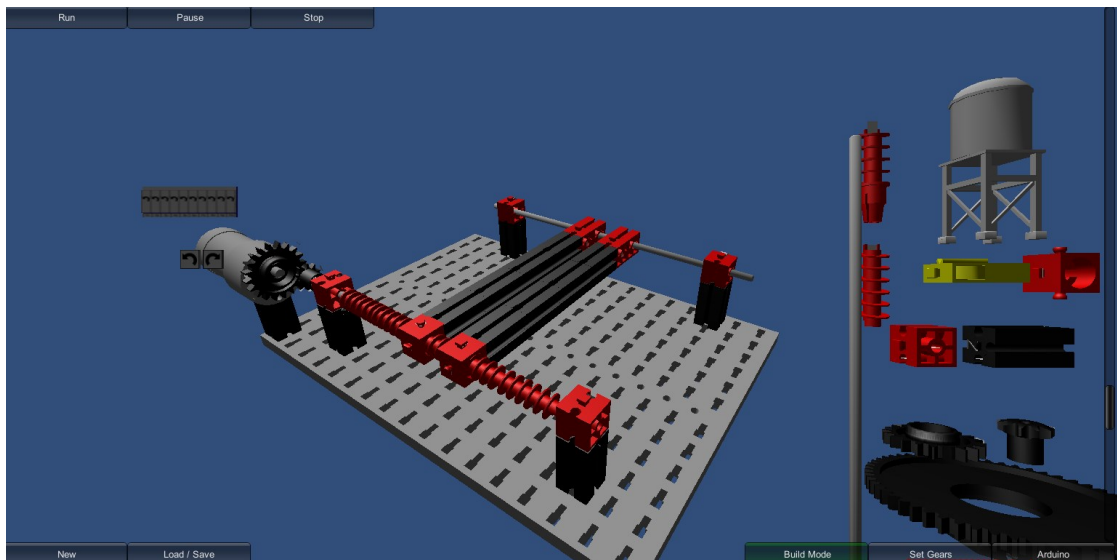
### First prototypical Iteration



**Figure 4.5:** Placing the first motor. The buttons for rotation or defining a motion control flow are immediately available.

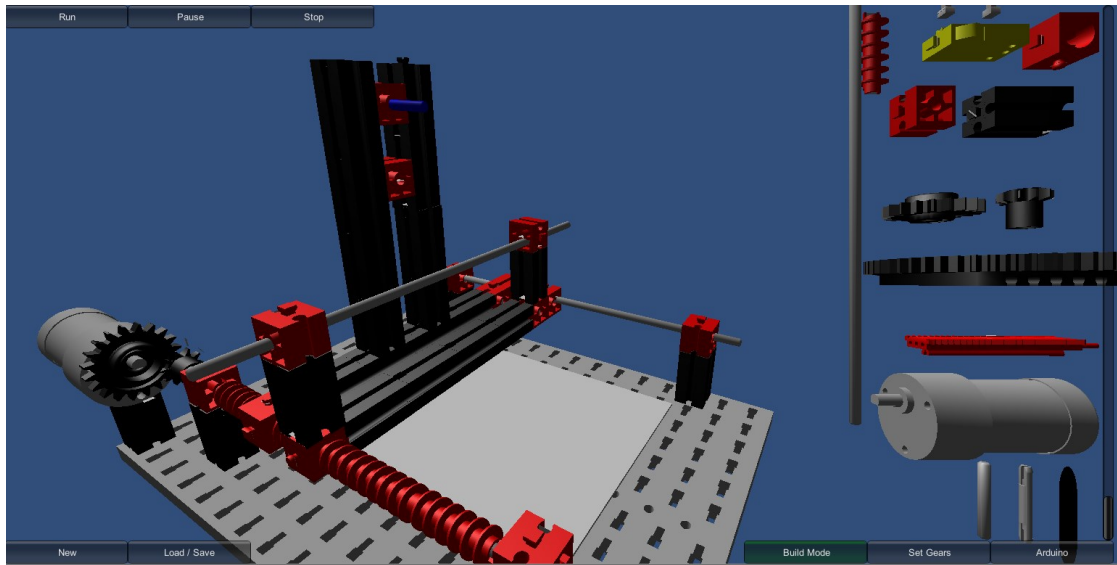


**Figure 4.6:** Screw threads are connected to the motor via a gear wheel.

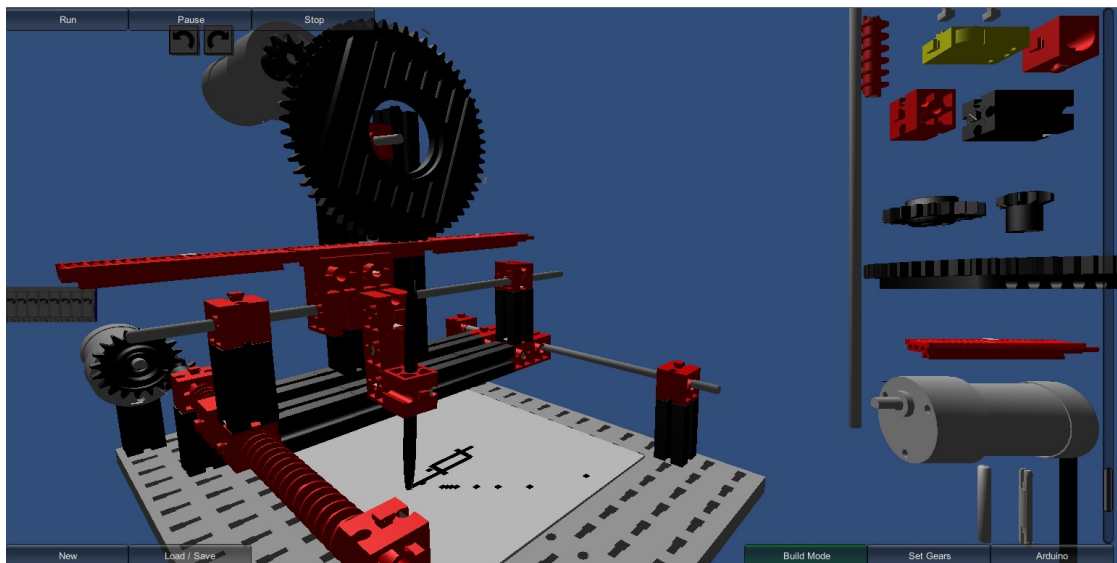


**Figure 4.7:** Base plate which is movable in one axis is placed.

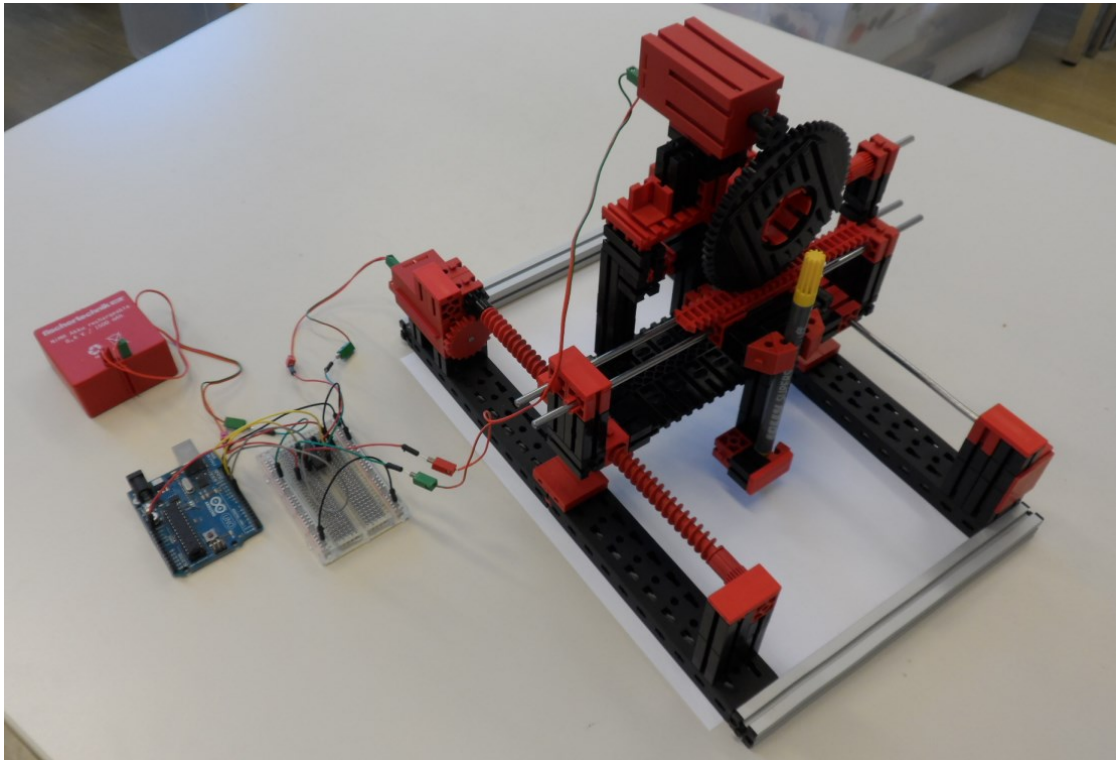




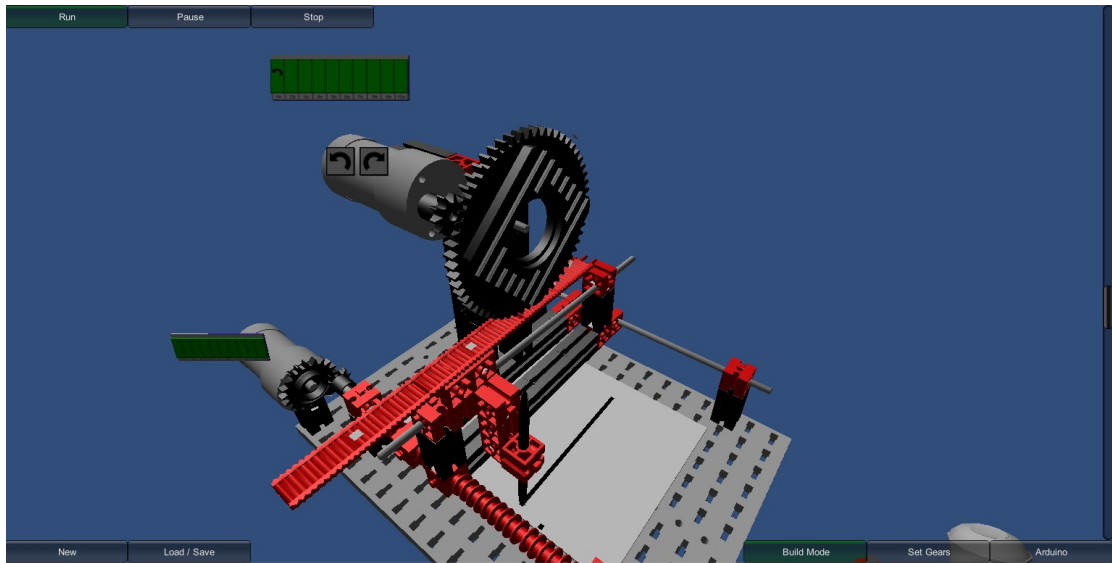
**Figure 4.8:** Preparing for the second motor and horizontal gear rails.



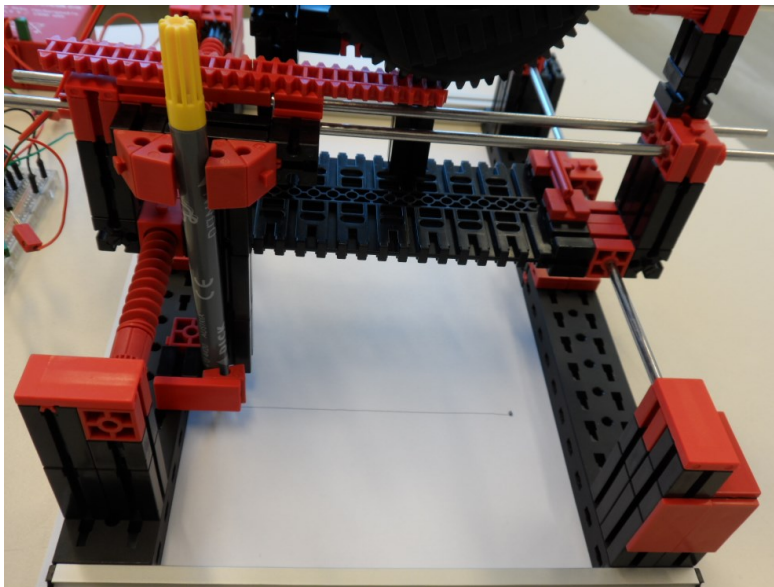
**Figure 4.9:** Horizontal gear rails where the pen is attached are placed. To lower the speed, the motor drives the gear rail via a gear wheel with a large diameter. This picture shows a first prototypical plot.



**Figure 4.10:** The red block is the power source (accumulator) for the motors. The blue electronic circuit board is an Arduino Uno which server as the communication interface bewteen the virtual simulation environment (PC) and the physical prototype (two motors). Its provided power is too low for rotating the motors. Therefore the accumulator powers the motors via a H-Bridge integrated circuit [27]



**Figure 4.11:** In a first try the motor on the top is rotated for one second counterclockwise, which results in a straight line. The problem is that the transmission does not lower enough the speed of the horizontal gear real. For the rooftop (diagonal lines) it is required that the motor can rotate two second in one direction.

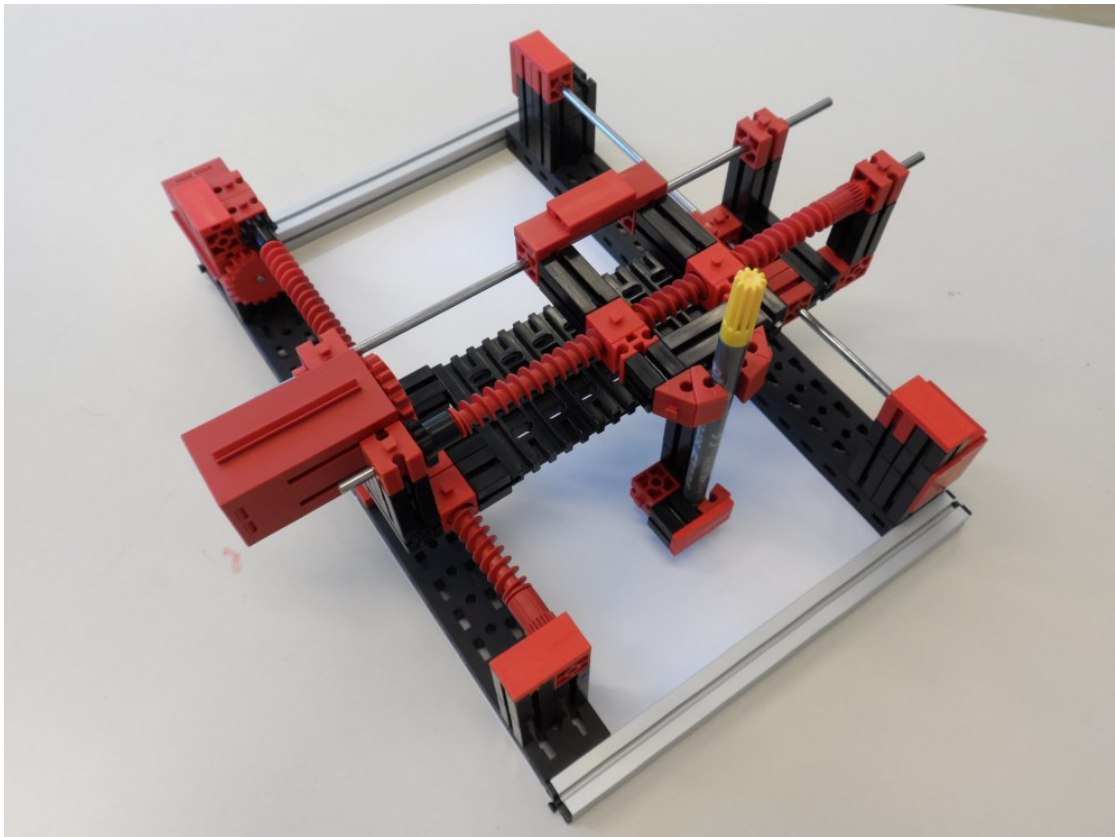


**Figure 4.12:** The physical 2D plotter shows the same problematical result as the virtual 2D plotter. Therefore a second design iteration for the plotter is required.

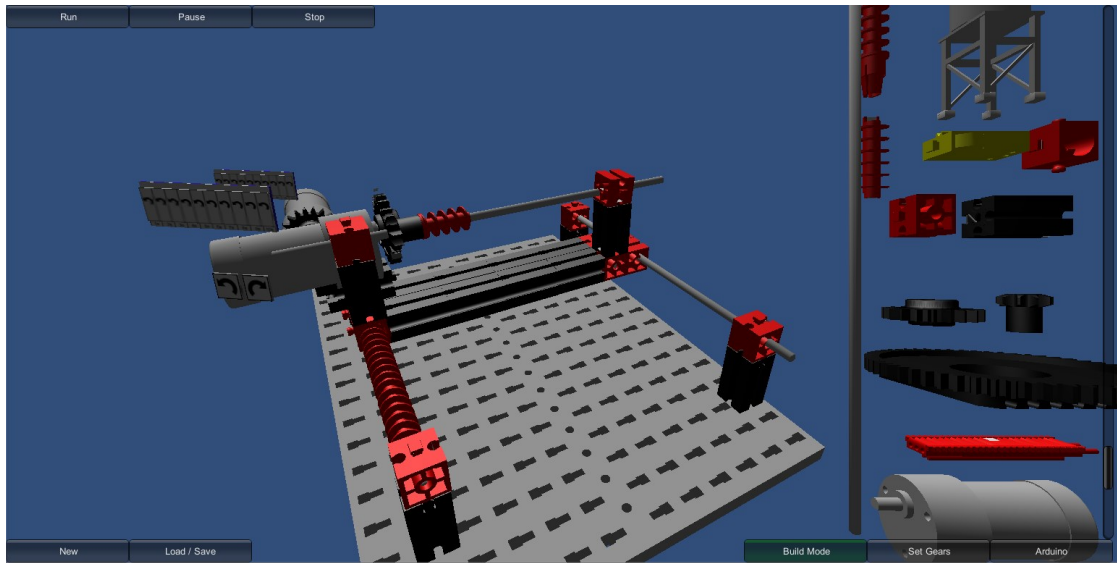
**Summary of the first iteration:** The time required for the virtual prototype of the 2D plotter were 30 minutes. Time delays occurred for estimating the right slots for placing two gear wheels next to each other.

For the physical 2D plotter with Fischertechnik 60 minutes of time were invested. Building the plotter stable and solid took time which is not required in the virtual environment. Fixing the pen took also more time than estimated. The clue is to have the pen not fixed too tight. The pressure of the pen on the paper results from the pens weight.

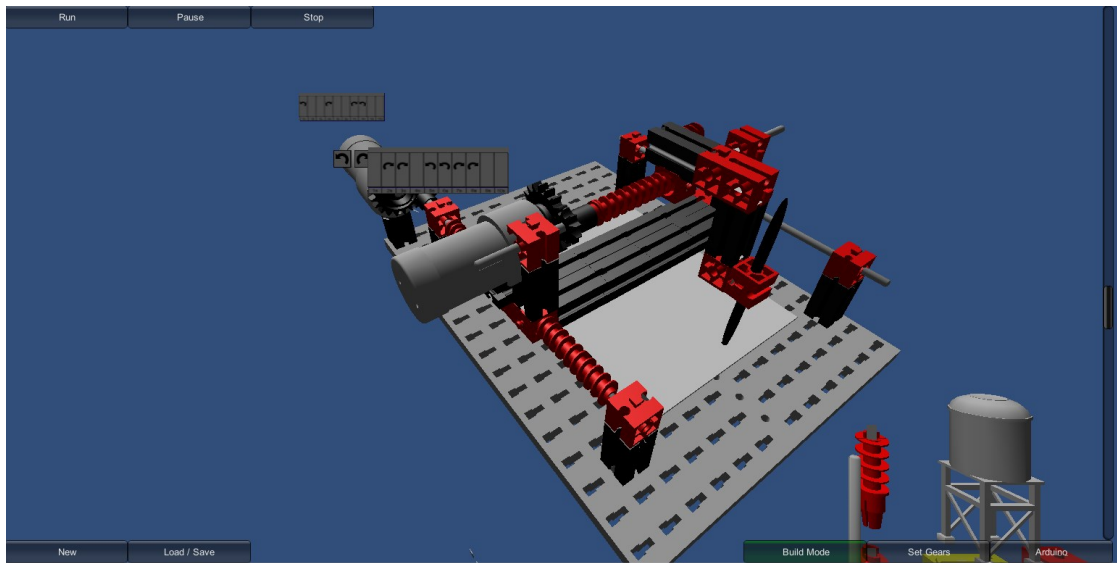
## Second prototypical Iteration



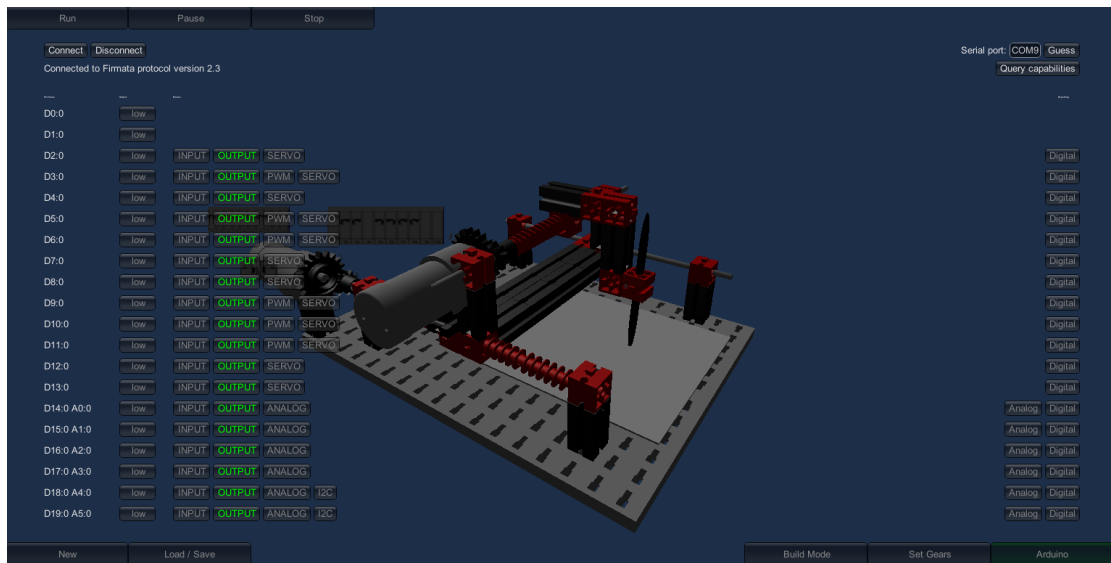
**Figure 4.13:** The transmission with horizontal screw thread worked fine in the first iteration. Therefore also for the motor on the top a screw thread transmission is attached.



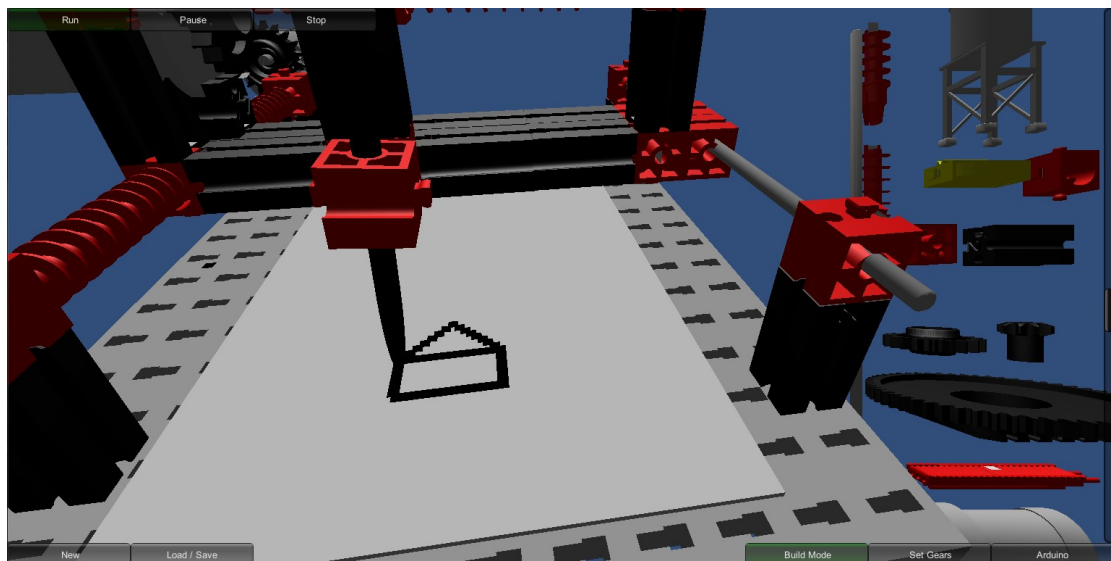
**Figure 4.14:** Most of the construction on the top is removed. The engine is now placed on the side and its attached gear wheel drives a screw thread.



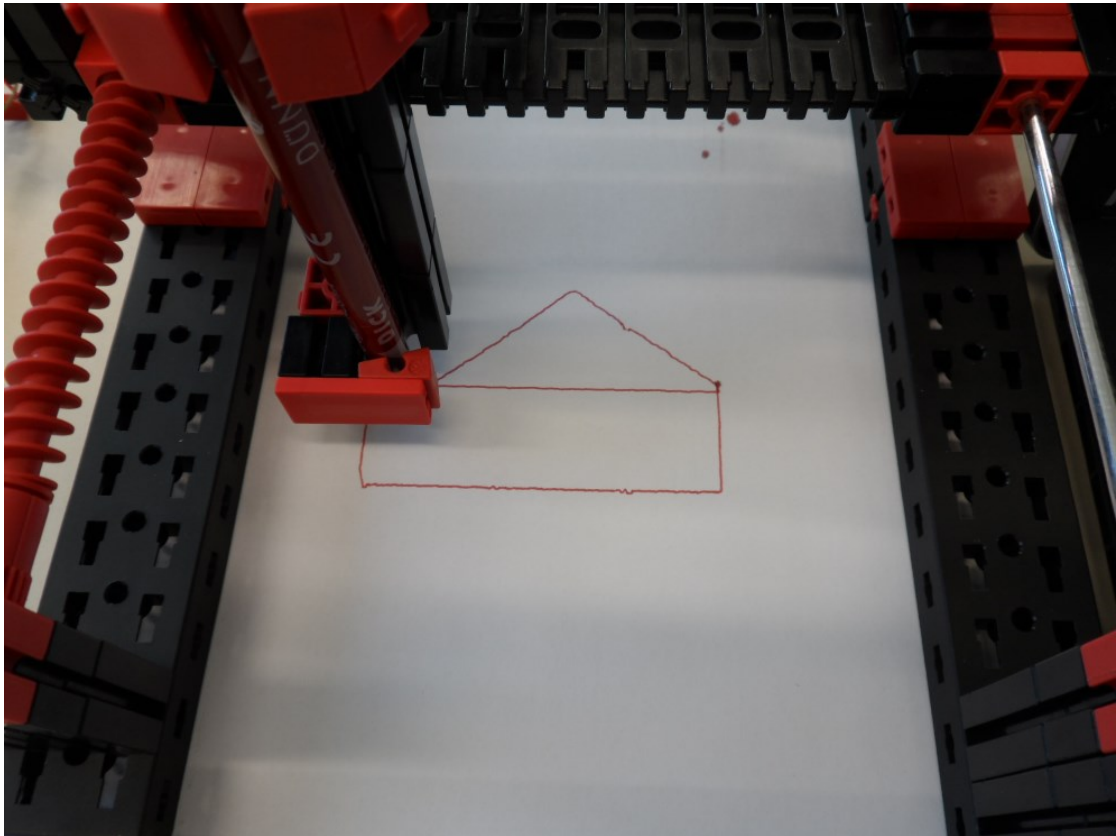
**Figure 4.15:** On the screw thread a construction for the pen is placed. Also the control flow is already defined for the two motors.



**Figure 4.16:** This screen depicts the connection interface to the Arduino Uno board. For each pin various configuration exists (e.g. input, output, digital, analog, value).



**Figure 4.17:** Starting the control flow results in the desired result, the two dimensional house.



**Figure 4.18:** The physical 2D plotter, connected to the virtual 2D plotter via the Arduino Uno board, plots also the desired house. But the result is not as accurate as in the virtual simulation. The mechanical resistance of the construction lowered the speed of one engine.

**Summary of the second iteration:** Removing pieces from the virtual prototype and rebuild the top part of the plotter took 15 minutes. Rebuilding the physical 2D plotter took also 15 minutes.

The task to plot a schematic house was achieved for the virtual and the physical prototype.

## 4.2 Comparison with Related Work

		Prototyping Environments								
		Performance Rating								
		Good	Average	Poor	Not supported					
Category	Criteria	Fischertechnik	Fischertechnik Designer	Creo Parametric Software	Phidgets	d.tools	Phybots	LEGO M. NXT	MRDS	Fischertechnik Simulator
Prototyping	Prototyping process	Good	Good	Good	Good	Good	Good	Good	Good	Good
	Prototyping of mechatronic systems	Good	Poor	Good	Good	Good	Poor	Good	Good	Good
	Virtual prototyping	Not supported	Average	Good	Average	Not supported	Not supported	Not supported	Not supported	Good
	Physical prototyping	Good	Not supported	Not supported	Good	Good	Good	Good	Average	Good
	Visual authoring	Good	Poor	Average	Not supported	Average	Not supported	Not supported	Not supported	Good
	3D modelling	Not supported	Average	Average	Not supported	Not supported	Not supported	Not supported	Average	Good
	3D control	Not supported	Poor	Average	Not supported	Not supported	Not supported	Not supported	Poor	Good
	2D control	Good	Average	Average	Not supported	Good	Not supported	Good	Good	Not supported
	Feedback system (debugging)	Good	Not supported	Not supported	Not supported	Good	Not supported	Not supported	Not supported	Poor
	Real-time behaviour	Good	Not supported	Good	Not supported	Not supported	Not supported	Not supported	Not supported	Good
	Usability	Good	Average	Average	Good	Good	Average	Good	Average	Good
	Understandability	Good	Average	Average	Good	Good	Good	Good	Average	Good
	Operating options	Good	Average	Average	Good	Good	Good	Good	Average	Good
	Attractiveness	Average	Good	Good	Good	Good	Good	Good	Good	Average
Technical	Interfaces of the System Architecture	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported	Good
	Hardware	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported	Good
	Complexity	low	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported	low
	Extendability	Not supported	Not supported	Not supported	Not supported	Not supported	Poor	Not supported	Not supported	Average
	Portability	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported	Good
	Software	Not supported	Average	Average	Not supported	Not supported	Not supported	Not supported	Not supported	Good
	Complexity	low	Not supported	high	high	high	high	low	high	Average
Extendability	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported	Average	
Portability	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported	Good	
Economic	Price (€)	400	120	200/Y	230	80	50	500	400	250
	Availability	Good	Good	Good	Good	Good	Good	Good	Good	Good
	Popularity (downloads/citations)	Good	Average	Good	Good	Average	Poor	Good	Good	Not supported
	Support	Good	Good	Good	Good	Good	Good	Good	Poor	Good

Figure 4.19: Comparison with related work.

Figure 4.19 is based on the evaluated criteria figure from section 2.4. It is enriched with the new designed and developed rapid prototyping environment *Fischertechnik Simulator*. The *prototyping process* is well supported by the option of immediately letting motors rotate. Therefore also the *prototyping of mechatronic systems* is well supported. Kinematic motions can be initiated by one click on the motors rotation buttons (3.2). The hierarchical representation is recognized by the virtual part of the *Fischertechnik Simulator* automatically (3.2). In the virtual simulation environment *Virtual prototyping* is provided by the feature of intuitively build and control virtual applications. These applications can be saved and loaded for later continuation of the virtual



prototype. It is only one step from the *virtual prototyping* to the *physical prototyping*. If the virtual and real environment is connected via the arduino board, the physical prototype operates automatically the same as the virtual prototype (3.2). The *visual authoring* tool does provide 3D modeling and 3D control of the virtual prototype. A 2D control is not implemented. The *Fischertechnik Simulator* does not have a fully implemented feedback system for debugging. Devices that are plugged on or off are not detected automatically. The *operating options* are clear to the user because the virtual environment is similar to the real environment. Only the graphical design is not state of the art. Some parts of the environment are still in a prototypical state. Connecting hardware is straight forward, therefore the *hardware* complexity is very low. But this has the side effect that the hardware is not very well extendable. To do so, new program code has to be written and additional electronic design have to be done. The *software* can only be extended by new program lines. It does not feature software services. The price of 250€ is about the fischertechnik hardware price without the control kit, but including a arduino set. Right now, this presented development kit is in beta, and therefore not yet available. It is planned to further develop the tool and also provide regular updates after a release of the *Fischertechnik Simulator*.

## Summary and future work

The goal of this thesis was to come up with various environments that feature rapid prototyping of mechatronic applications. Therefore the focus was set on virtual environments for developing and simulating mechatronic applications.

In the *Introduction* [1] mechatronic systems and applications from the manufacturing industries are given which implement software, electronics and mechanical hardware. The development of such systems is highly time consuming and costly. Also, people have to learn to operate with such complex industry applications. The goal of saving time and money leads to the requirement of having an environment that allows to develop prototypical mechatronic applications rapidly.

In the section *Survey of Prototyping Approaches* first, a criteria catalog for evaluating rapid prototyping environments is given and second, various rapid prototyping environments are presented. The presented environments have their focus in different areas. Fischertechnik and Lego Mindstorms NXT are visually programmable customer construction sets with actuators and sensors. Arduino is an programmable circuit board where actuators and sensors can be connected. Phidgets and d.tools are environments for rapidly developing physical interfaces. Their hardware and software model focuses on standardization for extendability and reuse. Phybots provides a programmable robotic development environment. Microsoft Robotics Developer Studio features a complex virtual simulation and real control of robotic applications. Applying the criteria catalog on these environments results in two findings: First, environments which feature easy to use rapid prototyping are lacking either the simulation part or the the implementation part. And second, environments which feature a virtual simulation and real mechatronic control are confronting the user with too much complexity.

Therefore the suggested solution is to come up with a new environment that covers virtual simulation and real control of mechatronic applications in a rapid prototyping process. This newly designed and developed environment is called *Fischertechnik Simulator* and allows to construct fischertechnik applications in a 3D virtual environment. The mechatronic behavior

of fischertechnik- actuators and sensors can be intuitively defined in the 3D environment. This defined mechatronic behavior can be transmitted to an analogue real fischertechnik construction via an arduino board. This was showed by designing a functional 2D plotter which plots a schematic house in the simulation of the 2D plotter and the corresponding physical implementation of the 2D plotter.

The Fischertechnik Simulator compared with the other environments showed that the features of the Fischertechnik Simulator are fulfilling the requirements from the criteria catalog by offering less complexity to the user. Therefore the Fischertechnik Simulator enables to rapidly develop prototypes for industrial applications with low costs.

For future work the Fischertechnik Simulator provides a good implementation of a mechatronic design environment. Open features for future updates are:

- High power actuators (motors) requires an external power supply and additional control logic. An optimal solution would be to attach the additional control logic on the motor and hide it from the user.
- Provide analog data for actuators and sensors. In the current release only digital values (5 Volt or 0 Volt) are implemented.
- Implementing a visual programming language. For complex mechatronic applications the control via the motors timing diagram can be difficult. Therefore an improvement for the visual control logic would speed up the prototyping process.
- Extend for Lego Mindstorms NXT bricks. Still, only a small set of fischertechnik pieces is implemented. The environment can be enriched with various pieces from all kind of construction sets.

# Bibliography

- [1] 3D Systems, Inc. Stereolithography interface specification. <http://people.sc.fsu.edu/~jburkardt/data/stla/stla.html>. visited on 19.06.2013.
- [2] Simon Adler, Zein Salah, Rüdiger Mecke, Georg Rose, and Bernhard Preim. Overlay of patient-specific anatomical data for advanced navigation in surgery simulation. In *Proceedings of the First International Workshop on Digital Engineering, IWDE '10*, pages 52–58, New York, NY, USA, 2010. ACM.
- [3] John Baichtal. *Basic Robot Building With LEGO Mindstorms NXT 2.0*. Que Publishing, 2013.
- [4] J. Banks. *Handbook of Simulation: Principles, Methodology, Advances, Applications, and Practice*. Wiley-Interscience publication. Wiley, 1998.
- [5] bildr.org. High-power control: Arduino + n-channel mosfet. <http://bildr.org/2012/03/rfp30n061e-arduino/>, 2012. visited on 13.05.2013.
- [6] Bill Gates. A robot in every home. *Scientific American*, 01/2007, 2007.
- [7] Tomas BREZINA, Vladislav SINGULE, and Zdenek HADAS. Model based design of mechatronic systems at brno university of technology. (2), 2012.
- [8] J.S. Cepeda, L. Chaimowicz, and Rogelio Soto. Exploring microsoft robotics studio as a mechanism for service-oriented robotics. In *Robotics Symposium and Intelligent Robotic Meeting (LARS), 2010 Latin American*, pages 7–12, 2010.
- [9] Godtfred Kirk Christiansen. Toy building brick. <http://worldwide.espacenet.com>, 10 1961.
- [10] David Cook. *Robot Building for Beginners (Technology in Action)*. Apress, 2010.
- [11] J. Craig. Simulation-based robot cell design in adeptrapid. In *Robotics and Automation, 1997. Proceedings., 1997 IEEE International Conference on*, volume 4, pages 3214–3219 vol.4, 1997.
- [12] Antonino Gomes de Sa and Gabriel Zachmann. Virtual reality as a tool for verification of assembly and maintenance processes. *Computers and Graphics*, 23:389–403, 1999.

- [13] Department of Computer Science, University of Calgary. The phidget project gallery. <http://grouplab.cpsc.ucalgary.ca/phidgets/>. visited on 2013/03/27.
- [14] Collin Reisdorf Amon Millner Ed Baafi. Modkit micro: Programming for anything. <http://www.modk.it/>. visited on 13.05.2013.
- [15] Charith Fernando, Jan Rod, David Siren Eisner, and Mauricio Cordero. Ikimo: open entry-level robotics platform. In *ACM SIGGRAPH 2012 Posters*, SIGGRAPH '12, pages 41:1–41:1, New York, NY, USA, 2012. ACM.
- [16] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext transfer protocol – http/1.1, 1999.
- [17] Roy Thomas Fielding. *Architectural styles and the design of network-based software architectures*. PhD thesis, 2000.
- [18] W.P. Flanagan and R. Earnshaw. Meeting the future at the university of michigan media union. *Computer Graphics and Applications, IEEE*, 17(3):15–19, 1997.
- [19] Maxim Garber and Ming C. Lin. Constraint-based motion planning for virtual prototyping. In *Proceedings of the seventh ACM symposium on Solid modeling and applications, SMA '02*, pages 257–264, New York, NY, USA, 2002. ACM.
- [20] Ilya B Gartsev, Leng-Feng Lee, and Venkat N Krovi. A low-cost real-time mobile robot platform (aredubot) to support project-based learning in robotics & mechatronics. In *Proceedings of 2nd International Conference on Robotics in Education (RiE 2011)*, R. Stelzer and K. Jafarmadar, Eds. INNOC–Austrian Society for Innovative Computer Sciences, pages 117–124, 2011.
- [21] Michael Gasperi and Philippe Hurbain. *Extreme NXT: Extending the LEGO MIND-STORMS NXT to the next level*. Technology in action series. Apress, Berkeley and Calif, 2 edition, 2009.
- [22] J. Gausemeier, Qing Shen, and J. Bauch. A solution elements based cooperative assembly system for mechatronic virtual prototyping. In *Computer Supported Cooperative Work in Design, 2004. Proceedings. The 8th International Conference on*, volume 2, pages 487–491 Vol.2, 2004.
- [23] Saul Greenberg and Chester Fitchett. Phidgets: easy development of physical interfaces through physical widgets. In *Proceedings of the 14th annual ACM symposium on User interface software and technology, UIST '01*, pages 209–218, New York and NY and USA, 2001. ACM.
- [24] Product Development Research Group. How new products are made. <http://www.bth.se/ing/pi.nsf/pages/pd>. visited on 25.08.2013.

- [25] J.O. Hamblen and G.M.E. van Bekkum. An embedded systems laboratory to support rapid prototyping of robotics and the internet of things. *Education, IEEE Transactions on*, 56(1):121–128, 2013.
- [26] Björn Hartmann, Scott R. Klemmer, Michael Bernstein, Leith Abdulla, Brandon Burr, Avi Robinson-Mosher, and Jennifer Gee. Reflective physical prototyping through integrated design, test, and analysis. In *Proceedings of the 19th annual ACM symposium on User interface software and technology*, UIST '06, pages 299–308, New York and NY and USA, 2006. ACM.
- [27] Stefan Hermann. Motorsteuerung mit einem h-bridge ic. <http://www.arduino-tutorial.de/2010/06/motorsteuerung-mit-einem-h-bridge-ic/>. visited on 03.11.2013.
- [28] Dr. Rainer Hessmer. Building a self-balancing robot. <http://www.hessmer.org/blog/2010/07/>. visited on 19.10.2013.
- [29] A.S Hornby and J. Turnbull. *Oxford Advanced Learner's Dictionary*. Oxford University Press España, S.A, 2010.
- [30] I2C -bus specification and user manual. I2c -bus specification and user manual. <http://www.nxp.com>, 2012.
- [31] IEC.
- [32] Anders Jönsson, Johan Wall, and Göran Broman. A virtual machine concept for real-time simulation of machine tool dynamics. *International Journal of Machine Tools and Manufacture*, 45(7–8):795 – 801, 2005.
- [33] Wesley M. Johnston, J. R. Paul Hanna, and Richard J. Millar. Advances in dataflow programming languages. *ACM Comput. Surv.*, 36(1):1–34, March 2004.
- [34] Jun Kato. Phybots: a toolkit for making robotic thing. <http://phybots.com>, 2013. visited on 28.03.2013.
- [35] Michael Kaminsky, Paul Dourish, W. Keith Edwards, Anthony LaMarca, Michael Salisbury, and Ian Smith. Sweetpea: Software tools for programmable embodied agents. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '99, pages 144–151, New York, NY, USA, 1999. ACM.
- [36] Jun Kato, Daisuke Sakamoto, and Takeo Igarashi. Phybots: a toolkit for making robotic things. In *Proceedings of the Designing Interactive Systems Conference*, DIS '12, pages 248–257, New York and NY and USA, 2012. ACM.
- [37] Yoshiharu Kato. Splish: A visual programming environment for arduino to accelerate physical computing experiences. In *Proceedings of the 2010 Eighth International Conference on Creating, Connecting and Collaborating through Computing*, C5 '10, pages 3–10, Washington and DC and USA, 2010. IEEE Computer Society.

- [38] M. Kerttula, M. Salmela, and M. Heikkinen. Virtual reality prototyping-a framework for the development of electronics and telecommunication products. In *Rapid System Prototyping, 1997. Shortening the Path from Specification to Prototype. Proceedings., 8th IEEE International Workshop on*, pages 2–11, 1997.
- [39] Pawel Kmiec. *The unofficial LEGO Technic builder's guide*. No Starch Press, 2012.
- [40] Jonathan Knudsen. *The Unofficial Guide to LEGO MINDSTORMS Robots*. O'Reilly Media, 1999.
- [41] A. Mackey, W.S Tulloch, and M. Krishnan. *Introducing .NET 4.5*. Expert's voice in .NET. Apress, 2012.
- [42] Michael Margolis. *Arduino cookbook*. O'Reilly, Beijing and Farnham [Dorset and England], 2011.
- [43] miscellaneous blog writers. brickipedia. <http://lego.wikia.com/>. visited on 23.05.2013.
- [44] P.R. Moore, J. Pu, H.C. Ng, C.B. Wong, S.K. Chong, X. Chen, J. Adolfsson, P. Olofsgård, and J.-O. Lundgren. Virtual engineering: an integrated approach to agile manufacturing machinery design and control. *Mechatronics*, 13(10):1105 – 1121, 2003. <ce:title>Mechatronics - a 12 year celebration</ce:title>.
- [45] Sara Morgan. *Programming Microsoft Robotics studio*. Microsoft Press, Redmond and Wash, 2008.
- [46] C.G Morris and Academic Press. *Academic Press Dictionary of Science and Technology*. Acad. Press, 1992.
- [47] Oracle and/or its affiliates. Javatm 2 platform standard edition 5.0 api specification. <http://docs.oracle.com/javase/1.5.0/docs/api/>, 2004. visited on 28.03.2013.
- [48] David J. Perdue and Laurens Valk. *The unofficial LEGO Mindstorms NXT 2.0 inventor's Guide*. No Starch, San Francisco and Calif, 2011.
- [49] Thomas Hirth Peter Elsner Peter Eyerer. *Handbuch Polymer Engineering*. (VDI-Buch) Chemische Technik Verfahrenstechnik. Springer, Berlin, 1 edition, 2007.
- [50] Promit Roy. Directx/xna phase out continues. <http://ventspace.wordpress.com/2013/01/30/directxxna-phase-out-continues/>. visited on 17.06.2013.
- [51] M. Schrage. *Serious Play: How the World's Best Companies Simulate to Innovate*. Innovation / Harvard Business School Press. Harvard Business School Press, 2000.
- [52] Zheng Shutao, Huang Qitao, Jin Jun, and Han JunWei. Flight simulator architecture development and implementation. In *Measuring Technology and Mechatronics Automation, 2009. ICMTMA '09. International Conference on*, volume 2, pages 230–233, 2009.

- [53] Software Engineering Institute. Glossary: Software architecture. <http://www.sei.cmu.edu/architecture/start/glossary/>. visited on 10.05.2013.
- [54] W.T. Tsai, Qian Huang, and Xin Sun. A collaborative service-oriented simulation framework with microsoft robotic studio. In *Simulation Symposium, 2008. ANSS 2008. 41st Annual*, pages 263–270, 2008.
- [55] G. Gary Wang. Definition and review of virtual prototyping. *Journal of Computing and Information Science in Engineering(Transactions of the ASME)*, 2(3):232–236, 2002.
- [56] Todd Zaki Warfel. *Prototyping: A practitioner's guide*. Rosenfeld Media, Brooklyn and N.Y, 2009.
- [57] Elecia White. *Making embedded systems*. O'Reilly Media, Inc., Sebastopol and CA, 1 edition, 2012.
- [58] Don Wilcher. *Learn Electronics with Arduino (Technology in Action)*. Apress, 2012.
- [59] H.C. Woithe and U. Kremer. An interactive slocum glider flight simulator. In *OCEANS 2010*, pages 1–10, 2010.
- [60] Matthew Wright and Adrian Freed, editors. *Open Sound Control: A New Protocol for Communicating with Sound Synthesizers*, Thessaloniki and Hellas, 1997. International Computer Music Association.
- [61] Xiaowen Xie, Beirong Zheng, and Xie Nan. Digital design experiments of mechanical product based on fischertechnik model. In *Mechanical and Electronics Engineering (ICMEE), 2010 2nd International Conference on*, volume 1, pages V1–237–V1–240, 2010.