Simbase: "Simulator Base Package"

Eva Rydén & Ralf Samuelsson

2002-11-22
Simbase: ”Simulator Base Package”

Examensarbete utfört i datorteknik vid Linköpings Tekniska Högskola, Campus Norrköping

Eva Rydén & Ralf Samuelsson

Handledare: Gull-Britt Isaksson
Examinator: Kent Axelsson & Jan Petersson

Norrköping den 22 november 2002
The purpose of this work is to providing a general set of base classes for building simulators, for example a robot system. Instead of every time building a new simulator from the beginning, you get the already completed components from a library. The library should in that way act as a springboard for development.

To do this work, there’s a lot of knowledge required in how a general robot system could look like. Therefore it was necessary to gather some information about robotics.

On the basis of what the information gave, two different user cases were built. From these user cases, the components were fetched, which was supposed to be included in the library.

The result of this work leads to a number of classes, which describes the components. A simple test program is also done, which handles the components. To get the classes more useful, the components must be broken down in much smaller parts. The operations on the classes should also be more realistic than what has been achieved in this work.
Abstract

The purpose of this work is to providing a general set of base classes for building simulators, for example a robot system. Instead of every time building a new simulator from the beginning, you get the already completed components from a library. The library should in that way act as a springboard for development.

To do this work, there’s a lot of knowledge required in how a general robot system could look like. Therefore it was necessary to gather some information about robotics.

On the basis of what the information gave, two different user cases were built. From these user cases, the components were fetched, which were supposed to be included in the library.

The result of this work leads to a number of classes, which describes the components. A simple test program is also done, which handles the components. To get the classes more useful, the components must be broken down in much smaller parts. The operations on the classes should also be more realistic than what has been achieved in this work.
Preface

This work has been performed at Department of Science and Technology in Norrköping, Linköping University and leads to a Bachelor of science in Computer and Electrical Engineering. The work is an obligatoric part in the education.

We would like to thank the company Fyrplus and the people at their local office in Linköping, for their help and for oppertunity to do this work.

We would also like to thank Kent Axelsson and Jan Petersson, whom has been our examiners during this work.
Table of contents

1 Introduction ..............................................................................................................1
   1.1 Background .......................................................................................................1
   1.2 Project purpose ..............................................................................................2
   1.3 Project goals ....................................................................................................2
   1.4 Report structure ..............................................................................................3

2 Robot systems .......................................................................................................5
   2.1 Application areas ............................................................................................5
   2.2 Complex systems ............................................................................................5
      2.2.1 Manipulator ..............................................................................................6
      2.2.2 Control System .......................................................................................7
      2.2.3 Programming ............................................................................................7
      2.2.4 End-effector ..............................................................................................7
      2.2.5 Sensors .....................................................................................................7
      2.2.6 Peripheral equipment ...............................................................................8

3 Simulation ............................................................................................................9

4 Object-oriented program development ..........................................................11
   4.1 Object-oriented analysis .................................................................................11
   4.2 Object-oriented design ...................................................................................11
   4.3 Object-oriented programming ........................................................................12
   4.4 What is an object? ..........................................................................................13
   4.5 Classes ............................................................................................................13
   4.6 Relations between classes and objects .........................................................14
   4.7 Sequence diagram ..........................................................................................14
List of figures

1. Figure in how a simulation might look like........................................... 2
2. A robot system with sub systems and its mutually relations............... 6
3. A manipulator with its central parts .................................................... 7
4. The object coordinates with conditions, operations and identity ........ 13
5. Figure of case one- actuator closed ...................................................... 16
6. Class diagram for case one ................................................................. 16
7. Figure of case one- visualisation ......................................................... 23
8. Figure of case two- the robots working area ....................................... 24
9. Class diagram for case two ................................................................. 25
10. Figure of case two- visualization......................................................... 44
1 Introduction

This chapter introduces the reader to the subject in this report. It will also describe the goal of the project.

1.1 Background

This work has been performed at the company Fyrplus in Linköping. The work is an obligatoric part in the education to a Bachelor of science in Computer and Electrical Engineering at Department of Science and Technology in Norrköping, Linköping University in 2002.

Fyrplus is an IT consulting company, which provide services and complete system solutions. Some of the branches that Fyrplus is working in are Security and Defence, Medical Industry, and Communication. The company are located in three cities, Karlstad, Linköping and Stockholm. They are about 60 employees and seven of them are located in Linköping.

There are many different types of simulator development, which can be carried out. To give a few examples there are development of flight simulators, simulation of small devices, and simulation of robots. In this project, the simulator is meant to be simulation of robots, which will be more like a electromechanical simulation system. A shorter introduction to why something should be simulated is described in chapter three.

Developments of applications that simulate some sort of behavior often require a lot of interaction with hardware and require complex design and implementation. Providing a general set of base classes for building simulators can act as a springboard for development. It also means that it’s possible to develop from an already tested platform regardless of what kind of simulator Fyrplus are building.

From without the created base classes, a simulator could be built up. A proposal of how it finally could look like are shown in the figure (Figure 1) on the next side. More explanations to the figure about how the simulator could be used are described in chapter five.
1.2 Project purpose

The purpose for this project is to find general classes designed in C++, which should be included in a library intended to work as a platform for simulator development.

1.3 Project goals

The main goal for this project is to provide a package of classes, which can be used as a platform for simulator development. This goal can be broken down as follows:
components should mirror real life components such as wires, motors and actuators
the package should provide a standard platform for simulator development at Fyrplus
the package should provide components for electrical and mechanical simulation
components should have a visual representation
the physical construction of the simulated device should be shown visually
the package should be implemented in an example simulation

1.4 Report structure

To find general components, which could be a part in a simulation system, the report starts with an example about how a general robot system might look like followed by some facts about simulation systems. The report also contains some facts about object-oriented program development. On the basis of that knowledge, two different user cases are represented, which includes the components. The work method, which has been used for both cases are the Unified Modeling Language (UML). UML is a specific method how to work with object-oriented program development, which is described in chapter four. The report completes with results, discussion and future work.

In the report there are some words of italic type. These are robotic terms, which are explained in a dictionary (Appendix 1- Dictionary). The references, which has been used is marked with angle brackets ([ ]).
2 Robot systems

This chapter should give the reader an overview in how a general robot system with its different parts could look like.

2.1 Application areas

In the first place you might identify a robot system with a mechanical arm, which performs the physical movement during the work. An automatization of a manufacturing process (without a mechanical arm), could also be seen as a robot system. However, there are many different application areas of use for robot systems. Apart from industrial applications there are others, for example in:

- space industry
- dangerous environments for humans
- medical services and rehabilitations
- education

[1]

2.2 Complex systems

In a more complex system there are many parts of systems that should cooperate to get the robot system to perform a task program. A robot system is generally built up in sub systems, which describes later in this chapter. The sub systems with its mutual relations could schematic be described as in figure 2. The arrows in the figure explains the relations between the sub systems. For example, the information about the task that should be performed, goes through all sub systems. The sub systems instead, only have a relation to the next following sub system in the schematic. [1]
2.2.1 Manipulator

The manipulator system could be divided into three different parts, the carrier, the arm and the wrist (see figure 3).

The carrier transport the robot to the workstation where it should do a specific task. The arm performs the position and movements in three dimensions for the end-effector during the operation. The wrist is used for orientation of the end-effector, which is attached to the wrist. The end-effector could be a 

*gripper* or a tool, for example a welding gun. To steer and decide the manipulators position and movements it is necessary to use a sensor, which measures position and velocity.

In both cases it is important to define the movement of the TCP (Tool Centre Point) in a coordinate system referred to the room.[1]
2.2.2 Control System

To perform different types of task program, the robot needs a control system. This system manages the power steering of the driving tool in the manipulator system. It also makes plans for, and performs defined movements and interpolations in the coordinate system.[1]

2.2.3 Programming

Programming of a robot means in the first place how to learn or define a specific operation. The most common method is to use a teach pendant which is connected to the control system. Another method is to physically grab the robot and move it through the desired action. [1]

2.2.4 End-effector

With an end-effector means all the units and systems that are attached on the robot's farthest link. The end-effector is the part that works directly with, or in the process. [1]

2.2.5 Sensors

With help from sensors could the robot system feel the condition and for the assignment essential process variables. The system could on the basis of this knowledge, suit its movements and working process in a changeable surroundings.[1]
2.2.6 Peripheral equipment

The most critical part, financially and technically, during the construction of a robot station is the peripheral equipment. That is due to the fact that the equipment usually is constructed in a few examples, while the robot under this connection could be seen as a mass-produced machine.

With peripheral equipment means all the equipment (except for the robot), which is included in the robot system to perform the desired action. The peripheral equipment could consist of:

- sensors
- fixtures
- machine equipment
- control- or computer equipments

[1]
3 Simulation

In this project you can say that the simulation system is a program to test an electromechanical system, for example a robot system. Much time and money could be spared by simulating the modelling of objects and equipments, which includes in a robot’s working area.

Simulation system is used to test different solution principles in a general application concerning to accessibility, cycle times or the flow of the material. Through this procedure it becomes possible in beforehand to control a program function. First and foremost it is logic and movements that are important to perform when a program is tested. With control of the movements it is mainly the working area and collision that is of the greatest interest. [1], [3]
4 Object-oriented program development

Object-orientation is the dominating technique of structure for system- and program development. The object-oriented technique creates (software) systems from program components (called classes), which is modelling the reality behind the system in a better way. This technique also make it easier to change and recycle. The work to create larger program systems will mostly consists of design, recycling and assembling of program components- instead of each time building the system all from scratch. The object-oriented program development is usually divided into three steps, analysis, design and programming.[4], [5]

4.1 Object-oriented analysis

The first step in the analysis is to understand and establish what the becoming program should do. The aim is to be familiar with the problem, perceive the conditions and to do a first wide model of the program, which should be created. More concrete, it´s important to:

- find the objects, which should be a part in the model
- describe the operations on the object
- establish the relationship between the different objects

The object could be found by let every real thing in the programs surrounding be an object. What the objects performs, decides the operations. Depending on how the objects cooperate with each other, the relationships between them are decided. No computer system or technical solutions are considered in the analysis. Only the activity or the method of working at the system we are modelling, should be studied.

The object-oriented model is a way to describe a system or an activity in a model where you can model what is acting (the object), which qualities they have and how they are related. It´s also describes how they cooperate to supply the totally function of the system. [5]

4.2 Object-oriented design

After the analysis comes the design phase. It´s in this phase where the programming will be planned. Instead of thinking about WHAT should be done in
the program (as in the analysis phase), you should think about HOW the thing should be done.

In the design, you started from the object that has been found in the analysis phase, and add a lot of details, object for object. You also decide the operations on the objects and then the parameters to the operations will be decided.

Object-oriented analysis and design could sometimes be difficult to separate. Changeover from the analysis to the design happens continuously. The process is iterated, which means that you have to go back to the analysis when something in the design phase should be changed.[5]

4.3 Object-oriented programming

The goal with the programming phase is to implement the system, to realize the system into a executed program. A program should have following qualities:

- correct
- efficient
- reusable
- changeable

The first demand is that the program should be correct. The operations, which has been defined in earlier phases of the program development, should be performed faultless.

The program should also be efficient, it should use the resources of the computer system in a good way.

The program development could be simplified if already constructed components can be used. The software should in other words be reuseable. The component should be suitable in different conditions and it must also be general.

The changebility increases if the program is constructed in several independent separated modules. Each module should have a special, well defined task.[5]
4.4 What is an object?

An object is the answer of WHAT we handles in our system or model. The object has a representation that describes the objects present permission and they have operations that can be applied in different ways to manipulate the object.

To sum up, the object has (see figure 4):

- conditions (a representation in the way of data elements)
- operations (functions that effects other objects and their conditions)
- identity (a way to separate the object from other objects)

![Coordinates diagram](#)

Figure 4  The object coordinates with conditions, operations and identity.

4.5 Classes

A class describes a type of objects. Inside the class, it describes all the characterization that all the objects of this type has; how the object of this class should be represented, which operations that should be possible to act on the object.

The class is described once in a program and it´s possible to create (instance) objects of this class. The class represents the type, the objects represents a sort of this type.

An object is always of a given class. When a class is defined it´s possible to create a lot of objects of this class. The objects is detached from each other, an operation on an object affects only this object. Objects of the same class has the same operations.
An object-oriented system consists of a number of classes and objects and when the system is completed, the objects will cooperate with each other in different ways. [5]

4.6 Relations between classes and objects

When modelling a system in classes and objects, the objects and classes should be identified. The relations between the different objects must be established.

One type of relation is inheritance. Inheritance happens between classes and not between objects. When a new class should be defined, inherits the characteristics from another existing class. In this new class, only the differences to the existing class are described.

The existing class is called super class and the new one are called sub class (or derived class). A class could be a super class to a class and a sub class to another class.

Using inheritance is a way to reuse both modelling and code. Through describing common parts in a super class and then have several sub classes, a better structure and understanding for the system is given. In this way you avoid that the classes becomes to large and complex. [5]

4.7 Sequence diagram

When using object-oriented design in a system developing process, the diagram are the most clearly method to show how the different objects cooperate. One sort of diagram is the sequence diagram, which shows interaction between a lot of objects. Especially interesting is the order in time between messages, that has been exchanged between different objects, during a dynamic course of events. [6]
5 User cases

This chapter is the most important part, since most of the actual work is located here.

If an object-oriented working method is being used, you have to define the objects and classes on the basis of a problem, which best models the situation. Since the main goal of this work was to design a set of general classes for simulation development, with no particular simulator system in mind, there are two different user cases created (the programming code for these cases that are made in C++, are not located into this report).

The analyses started with some studies around robotics. Even a description of an available robot system was studied. The next step was to create two different scenarios, user case one and user case two, which are described below. The last step in the object-oriented analysis was to decide the objects with its operations. The only relationship that was established was a few super and sub classes (inheritance).

5.1 Case one

Describes scenario, classes, sequence diagram and visualization for case one.

5.1.1 Scenario for case one

This case concerns manufactoring of medicine in a pharmaceutical industry. One part of the manufactoring phase is a production line, where the pills should be sorted out one by one. This can be done by a narrowing on the line (see figure 5).

To avoid stocking, a sensor (sensor 1) feels the quantity of the pills, which should be sorted out in a narrowing. The sensor send a signal to an actuator, which will steer the pills out from the line. Another sensor (sensor 2) feels that the actuator should go back, so that the pills could pass through again.
5.1.2 Classes for case one

The classes we identified for this case are:

If a sensor detects that something has happened, it will tell the actuator to be either open or closed. When a sensor detect something a signal passes through a wire (which is necessary to establish a connection) and into the robot. The robot, which is very important and also handles the traffic, pass it through another wire and into the actuator. The actuator will either be open or closed, depending on what the sensor has detected.
5.1.3 Sequence diagram for case one

The sequence diagram shows interaction between the objects in case one. Each signal have a unique number, which are described under the diagram.

**Actuator in its conditions, open or closed**

Signals:

01. s1 send signal to w1_in
02. w1 transfer signal to Robot
03. Robot read signal from w1_in
04. Robot send signal to w1_out
05. w1 transfer signal to Actuator
06. s2 send signal to w2_in
07. w2 transfer signal to Robot
08. Robot read signal from w2_in
09. Robot send signal to w2_in
10. w2 transfer signal to Actuator
5.1.4 Description of classes and functions for case one

Each class has a specific table directly after its introduction. The first area includes the operations (functions) and the following area its condition (data element).

**Class Actuator**

The actuator steers the pills to be on or off the line

<table>
<thead>
<tr>
<th>Class Actuator</th>
</tr>
</thead>
<tbody>
<tr>
<td>+Actuator</td>
</tr>
<tr>
<td>+open : void</td>
</tr>
<tr>
<td>+close : void</td>
</tr>
<tr>
<td>+get object : char*</td>
</tr>
<tr>
<td>-object : char</td>
</tr>
</tbody>
</table>

**open ()**

Function: Open the actuator
Parameters: No parameters

**close ()**

Function: Close the actuator
Parameters: No parameters

**getObject ()**

Function: Gets the name in the protected member
Parameters: No parameters
Protected member: Keeps the name of the object, which is instanced of the class
Class Robot

Robot is a class that handles the communication between all the different objects in both cases.

<table>
<thead>
<tr>
<th>Class Robot</th>
</tr>
</thead>
<tbody>
<tr>
<td>+Robot</td>
</tr>
<tr>
<td>+connectInput : void</td>
</tr>
<tr>
<td>+connectOutput : void</td>
</tr>
<tr>
<td>+sendSignal : void</td>
</tr>
<tr>
<td>+readSignal : void</td>
</tr>
<tr>
<td>-object : char</td>
</tr>
</tbody>
</table>

**connectInput (Wire& w, int inputNumber)**

Function: Establish a connection to Robot input
Parameters: Wire& w, inputNumber - The wire we want to connect and to which number

**connectOutput (Wire& w, int outputNumber)**

Function: Establish a connection to Robot output
Parameters: Wire& w, outputNumber - The wire we want to connect and to which number

**readSignal (Wire& w)**

Function: Reads a signal into the robot
Parameters: Wire& w - The wire we want to read from

**sendSignal (Wire& w)**

Function: Sends a signal out from the Robot
Parameters: Wire& w - The wire we want to send to

Protected member: Keeps the name of the object, which is instanced of the class
**Class Sensor**

Sensor detect something

```
+Sensor
+get object : char

-object : char
```

**getObject ()**

Function: Gets the name in the protected member  
Parameters: No parameters  
Protected member: Keeps the name of the object, which is instanced of the class

**Class Wire (super class)**

Wire carries the signal between two objects

```
+Wire
+receiveSignal : void
+transferSignal : void
+connection : void
+get object : char

-object : char
```

**receiveSignal ()**

Function: Receives a signal from an object  
Parameters: No parameters
transferSignal ( )
Function: Transfers a signal from an object
Parameters: No parameters

collection (char a[ ] )
Function: Connects an object to the wire
Parameters: a - The name of the object, which will be connected

generateObject ( )
Function: Gets the name in the protected member
Parameters: No parameters
Protected member: Keeps the name of the object, which is instanced of the class

Class WireP (sub class)
WireP carries the signal between two objects

<table>
<thead>
<tr>
<th>Class WireP</th>
</tr>
</thead>
<tbody>
<tr>
<td>+WireP</td>
</tr>
<tr>
<td>+receiveSignalFrom : void</td>
</tr>
<tr>
<td>+transferSignalTo : void</td>
</tr>
</tbody>
</table>

receiveSignalFrom (Sensor& s)
Function: Receives a signal from a sensor
Parameters: Sensor& s - A specific sensor

transferSignalTo (char object[ ] )
Function: Transfers a signal from an object
Parameters: Object - The name of the object where the signal should be transferred to
Parameters: Object - The name of the object where the signal should be
5.1.5 Visualization for case one

The simulator is intended to work as follows (see figure 7):

You choose a component from the class library and place it in the window. Continue until you have all the parts that you find necessary. You can choose to look at a signal in one part of the system, or in the complete course of events.

![Diagram of FYRPLUS Case 1](image-url)

**Figure 7** Figure of case one- visualisation
5.2 Case two

Describes scenario, classes, sequence diagrams and visualization for case two.

5.2.1 Scenario for case two

This case concerns a robot cell, where a specific task should be done. A “thing” should be moved in three steps, from an input station further to a work station and finally to an output station. The “thing” is moved by a manipulator with a gripper. The work station performs a specific working process on the “thing”. Here should the thing be heated up to a decided temperature. To get the whole system to work, it has to be a great number of sensors involved.

Figure 8  Figure of case two- the robots working area
5.2.2 Classes for case two

The classes we identified for this case are:

Several components in case two has the same relations as in case one.

If a sensor detects that some “thing” is at the InputStation, it will tell the robot that a movement of the manipulator to InputStation is necessary. What happens is that when an object of the class coordinates is created, the object will be sent to the class Positioner. These two classes cooperate with the class Manipulator, which is the component that actually does the movement. When the manipulator arrives to InputStation, it is possible for the Gripper to grip the “thing”. The same procedure is followed in each movement of the manipulator.

There are a few classes that actually are related. One example is the class Station, which is a super class to the three sub classes InputStation, WorkStation and OutputStation. In this example, every station has a name, a position in room coordinates and also the possibility to grip something. Each of the sub classes also have a sensor connected to them. The reason for inheritance is that WorkStation can use an oven to heat something up, while InputStation and OutputStation only can grip and transport a “thing”.

Figure 9  Class diagram for case two
5.2.3 Sequence diagrams for case two

The sequence diagrams shows interaction between the objects in case two. Each signal have a unique number, which are described under its diagram.

**The manipulator moves to Inputstation to grip a thing**

Signals:
01.  s1 detect thing on InputStation
02.  send signal to w1
03.  transfer signal
04.  read signal
05.  place coordinate
06.  send signal to w1
07.  transfer signal
08.  move to InputStation
09.  grip thing
The manipulator moves to WorkStation to drop a thing

Signals:
10. s2 detect that thing is gripped
11. send signal to w2
12. transfer signal
13. read signal
14. place coordinate
15. send signal to w2
16. transfer signal
17. move to WorkStation
18. drop thing
A thing has arrived to the WorkStation and is ready to be heated up

Signals:
19. s3 detect thing to work
20. send signal to w3
21. transfer signal
22. read signal
23. send signal to w3
24. transfer signal
25. start temperature
**Thing is done, so the temperature must be stopped**

<table>
<thead>
<tr>
<th>Sensor</th>
<th>WireT</th>
<th>Robot</th>
<th>Coord</th>
<th>Pos</th>
<th>Manip</th>
<th>MechGrip</th>
<th>InpSt</th>
<th>WorkSt</th>
<th>OutpSt</th>
<th>Temp</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>27</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Signals:
26. s4 detect that thing is done!
27. send signal to w4
28. transfer signal
29. read signal
30. send signal to w4
31. transfer signal
32. stop temperature
Thing must reach the start position in WorkStation, to be gripped

Signals:
33. s5 detect that thing is out of oven
34. send signal to w5
35. transfer signal
36. read signal
37. place coordinate
38. send signal to w5
39. transfer signal
40. move to WorkStation
41. grip thing
The manipulator moves to OutputStation to drop thing

Signals:
42. s6 detect that thing is gone
43. send signal to w6
44. transfer signal
45. read signal
46. place coordinate
47. send signal to w6
48. transfer signal
49. move to OutputStation
50. drop thing
5.2.4 Description of classes and functions for case two

Each class (except for these classes that already has been introduced in case one has a specific table directly after its introduction. The first area includes the operations (functions) and the following area its condition (data element).

**Class Coordinates**

Gives the position in room coordinates.

<table>
<thead>
<tr>
<th>Class Coordinates</th>
</tr>
</thead>
<tbody>
<tr>
<td>+Coordinates</td>
</tr>
<tr>
<td>+setX : void</td>
</tr>
<tr>
<td>+setY : void</td>
</tr>
<tr>
<td>+setZ : void</td>
</tr>
<tr>
<td>+getX : double</td>
</tr>
<tr>
<td>+getY : double</td>
</tr>
<tr>
<td>+getZ : double</td>
</tr>
<tr>
<td>-x : double</td>
</tr>
<tr>
<td>-y : double</td>
</tr>
<tr>
<td>-z : double</td>
</tr>
</tbody>
</table>

**setX (double xCoord)**

Function: Set the position for x
Parameters: xCoord - a decimal number

**setY (double yCoord)**

Function: Set the position for y
Parameters: yCoord - a decimal number

**setZ (double zCoord)**

Function: Set the position for z
Parameters: zCoord - a decimal number
**GetX ( )**
Function: Get the value in the protected member x
Parameters: No parameters

**GetY ( )**
Function: Get the value in the protected member y
Parameters: No parameters

**GetZ ( )**
Function: Get the value in the protected member z
Parameters: No parameters

Protected members: Keeps the value in x, y and z

**Class Gripper (super class)**
The gripper is an end-effector designed for seizing and holding

<table>
<thead>
<tr>
<th>Class Gripper</th>
</tr>
</thead>
<tbody>
<tr>
<td>+Gripper</td>
</tr>
<tr>
<td>+grip : void</td>
</tr>
<tr>
<td>+drop : void</td>
</tr>
<tr>
<td>+getObject : char*</td>
</tr>
<tr>
<td>-object : char</td>
</tr>
</tbody>
</table>

**grip ( )**
Function: Makes the gripper to grip a thing
Parameters: No parameters
drop ( )
Function: Makes the gripper to drop a thing
Parameters: No parameters

getObject ( )
Function: Gets the name in the protected member
Parameters: No parameters
Protected member: Object - keeps the name of the object

**Class InputStation (sub class)**
A specific place in the working area

<table>
<thead>
<tr>
<th>Class InputStation</th>
</tr>
</thead>
<tbody>
<tr>
<td>+InputStation</td>
</tr>
<tr>
<td>+thingToMove : void</td>
</tr>
</tbody>
</table>

thingToMove (Sensor& s)
Function: Thing to move on inputstation
Parameters: s - the sensor, which detects that there is something to move

**Class MagneticalGripper**
This class is not used in case two. It’s only included to show how you could proceed the development of the different components in a system.
The magnetical gripper is an end-effector designed for lifting and holding
Class MagneticalGripper

+MagneticalGripper
+lock : void
+unlock : void

lock ()
Function: Makes the magnetical gripper to lift a thing
Parameters: No parameters

unlock ()
Function: Makes the magnetical gripper to drop a thing
Parameters: No parameters

Class Manipulator

Manipulator is the arm that physically moves in the robots working area

Class Manipulator

+Manipulator
+moveTo : void
+moveToHome : void
+getObject : char*

-object : char

moveTo (Positioner& p, Station& s)
Function: Make a transportation of the manipulator
Parameters: p and s - The position and the station where the manipulator should move to
**moveToHome** (Positioner& \( p \))

Function: Make a transportation of the manipulator to its starting point

Parameters: \( p \) - The position where the manipulator should move to

**getObject** ( )

Function: Gets the name in the protected member

Parameters: No parameters

Protected member: Object - keeps the name of the object

---

**Class MechanicalGripper**

The mechanical gripper is an end-effector designed for seizing and holding

<table>
<thead>
<tr>
<th>Class MechanicalGripper</th>
</tr>
</thead>
<tbody>
<tr>
<td>+MechanicalGripper</td>
</tr>
<tr>
<td>+gripThing : void</td>
</tr>
<tr>
<td>+dropThing : void</td>
</tr>
</tbody>
</table>

**gripThing** (Manipulator& \( m \))

Function: Makes the mechanical gripper to grip a thing

Parameters: \( m \) - The manipulator that the end-effector is attached to

**dropThing** (Manipulator& \( m \))

Function: Makes the mechanical gripper to drop a thing

Parameters: \( m \) - The manipulator that the end-effector is attached to

---

**Class OutputStation**

A specific place in the working area
outThingToMove (Sensor & s)

Function: Thing to move on the outputstation
Parameters: s - the sensor, which detects that there is something to move

Class Positioner

The positioner has the responsibility that the manipulator moves to a specific position.

posTo (Coordinates & c)

Function: The position where the manipulator should move to
Parameters: c - the coordinates to the position

getObject ()

Function: Gets the name in the protected member
Parameters: No parameters
Protected member: object - keeps the name of the object
Class Robot

Robot is a class that handles the communication between all the different objects in both cases.

<table>
<thead>
<tr>
<th>Class Robot</th>
</tr>
</thead>
<tbody>
<tr>
<td>+Robot</td>
</tr>
<tr>
<td>+connectInput : void</td>
</tr>
<tr>
<td>+connectOutput : void</td>
</tr>
<tr>
<td>+sendSignal : void</td>
</tr>
<tr>
<td>+readSignal : void</td>
</tr>
<tr>
<td>-object : char</td>
</tr>
</tbody>
</table>

**connectInput (Wire& w, int inputNumber)**

Function: Establish a connection to Robot input
Parameters: Wire& w, inputNumber - The wire we want to connect and to which number

**connectOutput (Wire& w, int outputNumber)**

Function: Establish a connection to Robot output
Parameters: Wire& w, outputNumber - The wire we want to connect and

**readSignal (Wire& w)**

Function: Reads a signal into the robot
Parameters: Wire& w - The wire we want to read from

**sendSignal (Wire& w)**

Function: Sends a signal out from the Robot
Parameters: Wire& w - The wire we want to send to

Protected member: Keeps the name of the object, which is instanced of the class
Class Sensor
Sensor detect something

<table>
<thead>
<tr>
<th>Class Sensor</th>
</tr>
</thead>
<tbody>
<tr>
<td>+Sensor</td>
</tr>
<tr>
<td>+getObject : char*</td>
</tr>
<tr>
<td>-object : char</td>
</tr>
</tbody>
</table>

ggetObject ( )
Function: Gets the name in the protected member
Parameters: No parameters
Protected member: Keeps the name of the object, which is instanced of the class

Class Station (super class)
A specific place in the working area

<table>
<thead>
<tr>
<th>Class Station</th>
</tr>
</thead>
<tbody>
<tr>
<td>+Station</td>
</tr>
<tr>
<td>+thingGripped : void</td>
</tr>
<tr>
<td>+getObject : char*</td>
</tr>
<tr>
<td>-object : char</td>
</tr>
</tbody>
</table>

thingGripped (Sensor& s)
Function: Detect that “thing” is gripped
Parameters: s - the sensor, which detects that “thing” is gripped

ggetObject ( )
Function: Gets the name in the protected member
Parameters: No parameters
Protected member: Object - keeps the name of the object

Class Temperature
A working process

<table>
<thead>
<tr>
<th>Class Temperature</th>
</tr>
</thead>
<tbody>
<tr>
<td>+Temperature</td>
</tr>
<tr>
<td>+startTempProcess : void</td>
</tr>
<tr>
<td>+stopTempProcess : void</td>
</tr>
<tr>
<td>-object : char</td>
</tr>
</tbody>
</table>

startTempProcess (Station& s)
Function: Start the temperature (working process)
Parameters: s - the station, which requests that the thing should be heated up

stopTempProcess (Station& s)
Function: Stop the temperature (working process)
Parameters: s - the station, which requests that the temperature should be stopped

Class VacuumGripper
This class is not used in case two. It’s only included to show how you could proceed the development of the different components in a system.
The vacuum gripper is an end-effector designed for lifting and holding
Class VacuumGripper

<table>
<thead>
<tr>
<th>MagneticalGripper</th>
<th>underpressure : void</th>
</tr>
</thead>
<tbody>
<tr>
<td>releaseUnderpressure : void</td>
<td></td>
</tr>
</tbody>
</table>

**underpressure ( )**

Function: Makes the vacuum gripper to grip a thing

Parameters: No parameters

**releaseUnderpressure ( )**

Function: Makes the vacuum gripper to drop a thing

Parameters: No parameters

**Class Wire (super class)**

Wire carries the signal between two objects

Class Wire

<table>
<thead>
<tr>
<th>Wire</th>
<th>receiveSignal : void</th>
</tr>
</thead>
<tbody>
<tr>
<td>transferSignal : void</td>
<td></td>
</tr>
<tr>
<td>connection : void</td>
<td></td>
</tr>
<tr>
<td>get object : char*</td>
<td></td>
</tr>
</tbody>
</table>

| -object : char |

**receiveSignal ( )**

Function: Receives a signal from an object

Parameters: No parameters
transferSignal ( )
Function: Transfers a signal from an object
Parameters: No parameters

collection (char a[ ] )
Function: Connects an object to the wire
Parameters: a - The name of the object, which will be connected

genericObject ( )
Function: Gets the name in the protected member
Parameters: No parameters
Protected member: Keeps the name of the object, which is instanced of the class

Class WireTemp
Carries the signal between two objects.

<table>
<thead>
<tr>
<th>Class WireTemp</th>
</tr>
</thead>
<tbody>
<tr>
<td>+WireTemp</td>
</tr>
<tr>
<td>+receiveSignalFrom : void</td>
</tr>
</tbody>
</table>

receiveSignalFrom (Station& s )
Function: Transfers a signal from a Station
Parameters: s - the station, which receives the signal
**Class WorkStation**

A specific place in the working area

<table>
<thead>
<tr>
<th>Class WorkStation</th>
</tr>
</thead>
<tbody>
<tr>
<td>+WorkStation</td>
</tr>
<tr>
<td>+workThingIn : void</td>
</tr>
<tr>
<td>+workThingDone : void</td>
</tr>
<tr>
<td>+workThingOut : void</td>
</tr>
<tr>
<td>+getObject : char*</td>
</tr>
</tbody>
</table>

**workThingIn (Sensor& s)**

Function: Thing has arrived to the workstation  
Parameters: s - the sensor, which detects that something has arrived to the workstation

**workThingDone (Sensor& s)**

Function: The work is done with the thing  
Parameters: s - the sensor, which detects that the work is done

**workThingOut (Sensor& s)**

Function: Thing is ready to leave the workstation  
Parameters: s - the sensor, which detects that thing is ready to leave

**getObject ()**

Function: Gets the name in the protected member  
Parameters: No parameters  
Protected member: Object - keeps the name of the object
5.2.5 Visualization for case two

The simulator is intended to work as follows (see figure 10):

You choose a component from the class library and place it in the window. Continue until you have all the parts that you find necessary. You can choose to look at a signal in one part of the system, or in the complete course of events. You can also see the position of TCP (Tool Centre Point). The coordinates give the actual position.

Figure 10 Figure of case two- visualization
6 Results

The goals set for this project was:

To provide a package of classes, which can be used as a platform for simulator development. This goal can be broken down as follows:

- components should mirror real life components such as wires, motors and actuators
- the package should provide a standard platform for simulator development at Fyrplus
- the package should provide components for electrical and mechanical simulation
- components should have a visual representation
- the physical construction of the simulated device should be shown visually
- the package should be implemented in an example simulation

The result of this work leads to a number of classes, which describes the components. The classes has then been tested in two separated user cases.

There are several things that is possible to do in the test program, such as instance the classes (create objects) or calling a class by sending a reference from another class. There are a few subclasses made to provide the use of sub- and superclasses. The functions in these classes might not be the best, but they are necessary to provide the benefit with inheritance.

To get an understanding about what this project could lead to, we have also been speculated in how a visual part could look like. What we have done is only a sketch of a simulator, so we are aware of that it’s a lot of work if these visual parts actually should work. These sketches are shown in chapter five, there we also show how the simulator could act. An explanation about how it could be used is also included.

To do this work we had to study a lot of theory about robotics and object-oriented systems development. During the work we also had an excellent time to practise the programming language C++ and we both thought that our knowledges in programming have increased. We have got an understanding about how to build up a system from the beginning by thinking in an object-oriented way.
7 Discussion

If a simulator should be done, it’s important to know that further more time and knowledge are required. Even though we have worked with the project for several weeks, it is still hard to understand how a simulator actually should look like, and work. On the other hand, we think that we now are quite familiar with the theory and the different parts that can be included in a robot system.

To have enough sufficient classes that works, it’s necessary to break down the components in a greater extent then we have done or have had the knowledge to do. For example the class “Workstation”, which consists of many parts, could be broken down into several classes.

It wasn’t easy to realise how much work that was necessary to complete all the goals. The hope was to complete them all, even if that has lead to a few, but well organised classes. The first weeks were necessary to learn something about the robotics and to understand which components that could be included in a robot system. Our limited knowledge in C++ also made that the program code took a big part of the time, which was set for this project.

If Fyrplus should proceed with this project, or offer the project once again, it’s desirable for the people involved, to having the opportunity to fulfil all the goals in the project. We knows that it’s not possible to create a simulator that work completely, in the time set for some student during their examination work. Maybe it would be better if a smaller user case is set up with a few well-organised classes. Then there might be a possibility to have a visual represention and a class library.
8 Future work

The main goal for this project was to provide a package of classes, which could be used as a platform for simulator development. This goal could be broken down as follows:

- components should mirror real life components such as wires, motors and actuators
- the package should provide a standard platform for simulator development at Fyr-plus
- the package should provide components for electrical and mechanical simulation
- components should have a visual representation
- the physical construction of the simulated device should be shown visually
- the package should be implemented in an example simulation

We didn’t have the time to fulfil all these goals during our examination work. It remains a lot of work before there is a simulator that actually works.

The first thing to do is to review all the existing components. They must be broken down in smaller parts that are more efficient and also mirror real life. It’s first and foremost then that the classes could be included in a library. The library should work as a platform for simulator development.

The components should also have a visual representation. In that case there are two sketches made in how they could work and be used. Hopefully these sketches could be guidance in the way to decide how it finally might look like. To solve the visual parts, we have no proposal since our knowledge within the subject is limited.
References

*The book considers the components and sub systems of the industrial robot.*

*The definition of an industrial robot, its mechanical parts, control system and peripheral equipments.*

*This book is essential reading for engineers developing robotic systems.*

*This book brings up the complete language of C++ - from the beginning to the most advanced constructions of the language.*

*The development of object-oriented programs in C++ describes the theory behind the object-oriented model.*

*An object-oriented system development model for process oriented business development.*
10 Appendix 1- Dictionary

End-effector: A tool, which is physically attached to the robot. It is with help from that tool, the robot perform its assignments.

Gripper: An end-effector designed for seizing and holding.

Manipulator: A machine, which purpose is to grasping and/or moving objects.

Task program: The set of motion and auxiliary are function instructions, which define the specific intended task of the robot system.

Teach pendant: A hand-held robot control terminal that provides a convenient means to move the robot, teach locations, and run robot programs. The movements are set by buttons or by a joystick.