TECHNISCHE UNIVERSITÄT WIEN
Vienna University of Technology

ACIN
AUTOMATION & CONTROL INSTITUTE
INSTITUT FÜR AUTOMATISIERUNGS- & REGELUNGSTECHNIK

# Simulation of LEGO® Mindstorms NXT Based Plants Using a Distributed Automation Framework

## DIPLOMARBEIT

Ausgeführt zum Zwecke der Erlangung des akademischen Grades eines

Diplom-Ingenieurs (Dipl.-Ing.)

unter der Leitung von

Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Markus Vincze
Dipl.-Ing. Dr.techn. Alois Zoitl

eingereicht an der

Technischen Universität Wien
Fakultät für Elektrotechnik und Informationstechnik
Institut für Automatisierungs- und Regelungstechnik

von

Bakk.techn. Pirker, Simon
Johann-Teufelgasse 39–47/6/12
1230 Wien
AUT

Wien, 2. Dezember 2010 _____ _____
(Unterschrift Verfasser) (Unterschrift Betreuer)

# Erklärung zur Verfassung der Arbeit

Simon Pirker, Johann-Teufelgasse 39-47/6/12, 1230 Wien, AUT „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.“

Wien, December 2, 2010,

# Vorwort

Das Verfassen dieser Arbeit, bedeutet den baldigen Abschluss meines Studiums. Es war eine Zeit mit vielen lehrreichen Erfahrungen. Dass ich diese Erfahrungen machen durfte, verdanke ich vor allem meinen Eltern, die mich bedingungslos unterstützt haben, und auch meinem Bruder, der mich überhaupt erst auf den Pfad der Informatik brachte. Außerdem möchte ich mich bei meiner Freundin bedanken, die mich moralisch sehr unterstützt hat. Zuguter letzt gebührt meinem Betreuer Dank, der seine Rolle exzellent ausgeübt hat, und mir viel nützliches Feedback gegeben hat.

Wien, December 2, 2010

# Abstract

To build a new factory, or to change the production chain can be a very cost intensive endeavor. Therefore a lot of interest exists in cost reducing methods. One way to reduce cost during the development phase is provided by the concept of the Digital Factory, where the production flow is simulated and evaluated in a realistic manner. The Digital Factory not only consists of a simulation part, but also of a control part, where the control applications can be tested. When those control applications are run on a real controller, and interact with the simulated factory, we can speak of Soft Commissioning®, which enables early error detection. With this early error detection, possible cost intensive changes in the factory can be prevented later on, and the design phase is being shortened, due to the flexible panning, resulting in earlier start up of the factory.

Within this thesis the question will be answered, whether a 3D game engine is suited well enough, to develop a distributed simulation framework, following the concept of the two tier model of the Digital Factory. For the creation of the Digital Factory, 3D models of LEGO® Mindstorms NXT sensors and actuators will be used, which are available from the LDraw library. For the development of the control part, 4DIAC-IDE will be used, and FORTE as the runtime environment.

Using the developed framework, own control applications can be created, which interact with the simulation part. That way the concept of the Digital Factory and Soft Commissioning® can be experienced by studetns.

In order to test the framework, two experiments are developed in this thesis. Those are a car wash station, and a robot, which follows a line. Next to presenting the results of the framework and the experiments, also the usage of the 3D game engine will be presented. The results show, that a 3D game engine is well suited to develop a framework of a Digital Factory, which can also be used for teaching purposes.

# Kurzzusammenfassung

Eine neue Fabrik zu bauen, oder auch nur den Produktionsablauf zu ändern ist ein sehr kostspieliges Unterfangen. Deshalb wird in kostenreduzierende Maßnahmen viel investiert. Ein Ansatz, um die Kosten schon während der Entwurfsphase zu senken, bietet die Digitale Fabrik. Dabei werden die Abläufe in der geplanten Fabrik realitätsgetreu simuliert, und ausgewertet. Eine Digitale Fabrik besteht aber nicht nur aus dem Simulationsteil, sondern auch aus einem Steuerungsteil, in dem Steuerungsprogramme getestet werden können. Wenn diese Steuerungsprogramme auf der echten Hardware ausgeführt werden, und damit der Simulationsteil gesteuert wird, spricht man von Soft-Commissioning®, wodurch Fehler frühzeitig erkannt werden können. Durch die frühen Tests werden spätere teure Änderungen an der Fabrik verhindert, und durch die flexible Planung wird die Planungsphase verkürzt, wodurch die Fabrik früher realisiert werden kann.

In dieser Arbeit wird die Frage behandelt, ob sich 3D Spiele Engines dazu eignen, ein Framework für verteilte Simulationen nach dem Konzept der Digitalen Fabrik zu entwicklen. Für die Darstellung der Digitalen Fabrik werden 3D Modelle der LEGO® Mindstorms NXT Sensoren und Aktuatoren verwendet. Die Steuerungsprogramme werden IEC 61499 konform mit der 4DIAC-IDE entwickelt, und in der FORTE Laufzeitumgebung ausgeführt.

Das entwickelte Framework bietet die Möglichkeit, über eine Schnittstelle eigene Steuerungsprogramme zu entwerfen, die mit dem Simulationsteil interagieren. Diese Möglichkeit ist vor allem für den Lehrbetrieb interessant, da es Studenten die Möglichkeit bietet, früh mit dem Konzept der Digitalen Fabrik, und des Soft-Commissioning® vertraut zu werden.

Zum Test des Frameworks werden in der Arbeit zwei Experimente entwickelt. Diese sind die Simulationen einer Autowaschanlage, und eines Roboters, der einer Linie folgt. Neben den Ergebnissen zu den Experimenten und zum Framework, wird auch der Umgang mit der 3D Engine selbst exemplarisch erläutert. Die Resultate zeigen, dass sich eine 3D Spiele Engine sehr gut dazu eignet, ein Framework zu entwickeln, das alle Anforderungen einer Digitale Fabrik erfüllt, und sich auch für den Einsatz im Lehrbetrieb eignet.

# Contents

# List of Figures

# 1 Introduction

Simulation in the automation industry has undergone a lot of research. One of the concepts that have been developed during that research is the Digital Factory. The Digital Factory is a virtual factory, which consists of a simulation of the real factory, and offers integration of the various planning and simulation processes. This concept has been used in professional simulation tools, which offer the benefit of early testing, while the factory is still in the design phase. This leads to Virtual Commissioning, the virtual launching of the factory, where design errors can be found, before the factory is actually being built. The Digital Factory is based on distributed simulation, which has mainly been applied in the military sector for simulation environments, but has also found its way into the automation industry. There the benefits of the distributed simulation contribute to accelerated execution times, and support interchangability of parts of the simulation with the real system.

For teaching purposes simulation is very useful, since in courses hardware might not be available for every student at all times. Simulation in courses offers:

- Reduced hardware expenses
- Enhanced availability
- Early contact with the concept of the Digital Factory
- Flexibility

For those reasons it would be a good idea to use simulation in courses at the university, but there are even more issues which could be handled with simulation. One problem arises at the hardware. Devices like Programmable logic Controllers (PLC) are available at Universities, and used in courses. Still, due to the cost, and size of such setups, it is inherent that the access to such systems for students is limited. Current projects try to increase the availability of hardware. One of them has the goal to make the LEGO® Mindstorms NXT platform compatible to the automation

standard IEC 61499[1]. Still in those projects also hardware is necessary, and even if it is cheaper and has a better availability, there are limits. Therefore the benefits of simulation could also be used in this section of hardware substitutes, bringing the expenses almost to zero, and making them available for any ordinary PC.

Another benefit of distributed simulation is, that already existing simulation tools can be embedded and connected to form bigger systems. An example can be found in the automation area, where the tools 4DIAC-RTE/FORTE[2] and FBRT[3] offer simulation of PLC hardware, and can be executed in an embedded system.

One very interesting feature which can be contained in the Digital Factory is Soft Commissioning®. This is the possibility to exchange parts of the simulation with the real system for early testing purposes. For example the sensors and actuators of the LEGO® Mindstorms NXT devices could be simulated, but controlled by a real PLC. Also the other way around would be possible where a PLC is being simulated within the LEGO® Mindstorms NXT controller brick, but the real sensors and actuators can be used. Using such combinations all features of Soft Commissioning® can be used on a low cost hardware platform.

The Digital Factory requires the simulation of the sensors and actuators to be as accurate as possible. The simulation should offer feedback for the user, in an obvious and realistic way. This leads to the necessity of visualization, since it offers direct feedback in an easy and understandable way. For realistic visualizations, speed is a key factor. A simulation that lags behind is of little use. 3D game engines offer all of those requisites. They are designed for fast and realistic display of virtual worlds.

Analyzing the ideas mentioned above, three fields of interest can be identified. First there is the Digital Factory, which concepts allow cost reduction when planning a factory, and the interaction of simulation and the real system. Second, the use of 3D game engines for simulation, and third the possible integration of simulation into the teaching process. Combining those fields is the goal of this thesis, and led to the following task description.

[1] A new family of standards within the IEC for Industrial Process Measurement and Control Systems (IPMCS)

[2] Modular IEC 61499 compliant Runtime Environment for small embedded devices 16/32*Bit*, implemented in C++

[3] Function Block Run Time- a run time environment for IEC 61499 applications developed by [9]

## 1.1 Task Description

The main question of this thesis is, whether 3D game engines can be used to form a simulation according to the concepts of the Digital Factory. The Digital Factory has undergone a lot of research, and it offers concepts, which can be used to simulate a factory. This concepts should be applied to 3D game engines, resulting in a distributed simulation framework. This framework should be designed in a way, so that it can be used in the teaching process. In the end, the developed system should be tested by implementing experiments, using the framework.

The requirements from the teaching point of view are the usage of an IEC 61499 compatible controller, and LEGO® Mindstorms NXT sensors and actuators. The simulation should be distributed, where the sensors and actuators should form one side, and the controller block should form the other side. The connection should be achieved through a communication network, so all parts can be executed on different Computers. Within this setup, it should be examined how well 3D game engines can be used for Digital Factory simulations.

## 1.2 Approach

From the task description some basic outline of the solution can be sketched. As depicted in Figure 1.1, an IDE (Integrated Development Environment) is needed in order to develop the simulation. From the development point of view, there are three parts, which can be developed independently, so the IDEs do not need to be the same. For the control application, the IDE is given to be the Framework for Distributed Industrial Automation and Control [2]. Using 4DIAC, the control application will be developed, and run in FORTE [9], which supports execution of applications developed with 4DIAC, as a runtime environment for IEC 61499 compatible applications.

For the other two IDEs, there are no restrictions, so they can be chosen freely for developing the simulation of the LEGO® hardware, and the communication network. For the simulation a 3D game engine will be selected, in order to provide the user interface and to simulate the physical attributes of the LEGO® hardware. The 3D models of the LEGO® hardware can be found in the library provided by LDraw [15]. The communication network will connect the two parts of the simulation.

In order to be used for teaching, the simulation has to be designed in a way, so

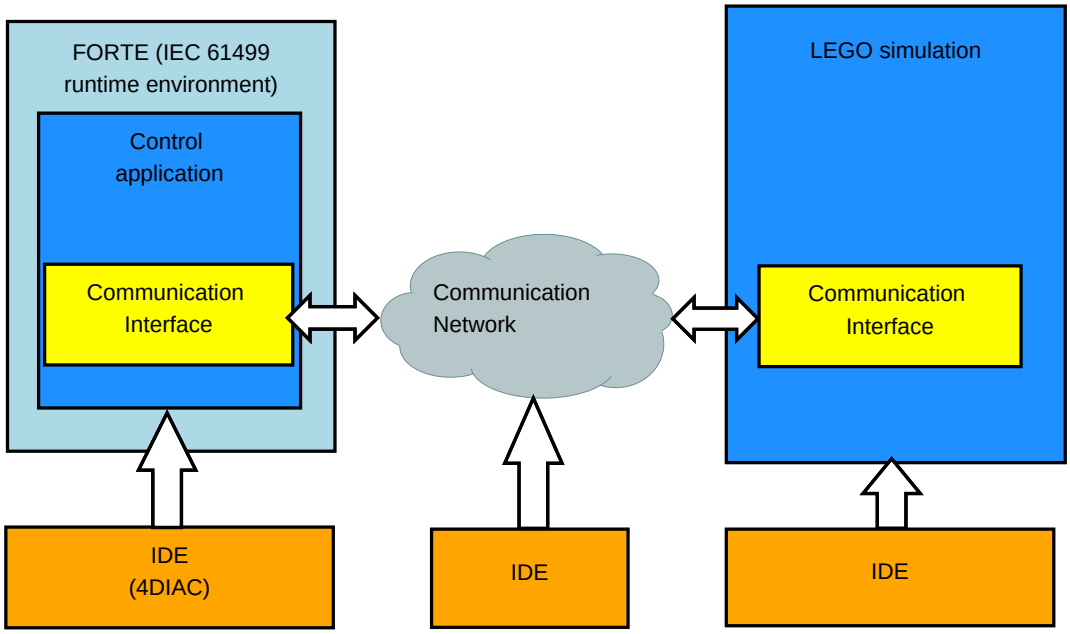


Figure 1.1: Main parts of the system

that students can create their own applications. Using the results of two experiments, two different systems will be designed. Those systems will consist of the simulated LEGO® Mindstorms NXT hardware, and the control application. The control applications can be replaced with other applications written by students. An interface will be provided to access the functions of the hardware simulation within the control application. This interface will be similar to the interface provided by the LEGO® Mindstorms NXT to access its sensors and actuators. [22]

## 1.3 Guideline Through the Work

In the next section, the *state of the art* will be described, providing the necessary information, on which this thesis is based. The section following *the state of the art* provides the *Concept* of the thesis, where different possible ways of the solution are compared and discussed. Out of those possible solutions, the most feasible is chosen, and its components will be discussed in more detail. This will lead to a description of the requirements, which have to be satisfied, and by which means they will be applied, resulting in some UML diagrams, describing the desired system. Also the two experiments will be planned, containing the basic system, which both experiments have in common, and the extended system, where the differences will

be stated.

In the section *Implementation* it will be presented, how the simulation was developed. The basic system of the simulation will be presented in detail, including the most important sections of the source code. Also the implementation of the communication network will be presented, and how the interface of the real LEGO® Mindstorms NXT controller will be mapped into the IEC 61499 Function Blocks.

The two experiments will be presented in the section *Discussion of results*. One of the experiments is a robot, which follows a line, and the other is a car wash station, where the brush has to follow the shape of a car. Also the quality of the simulation will be discussed, and whether a 3D game engine can be used for implementing a Digital Factory. The results will be compared to the task description, in order to find out, whether all the requirements have been satisfied.

The *Outlook* will give ideas how this thesis can be extended, and which parts could be carried on in the simulation, in order to provide more functionality and flexibility.

Finally in the section *Conclusion* the results of the thesis will be summarized.

# 2 State of the Art

Now that the problem has been outlined in the previous chapter, we will survey what solutions already exist. Some of the concepts, which will be presented in this chapter may be applicable to our solution of the problem. In the fields of interest, only relevant information will be presented, which will be needed in the rest of the thesis. An overview of the connections of the topics in this chapter is given in Figure 2.1.

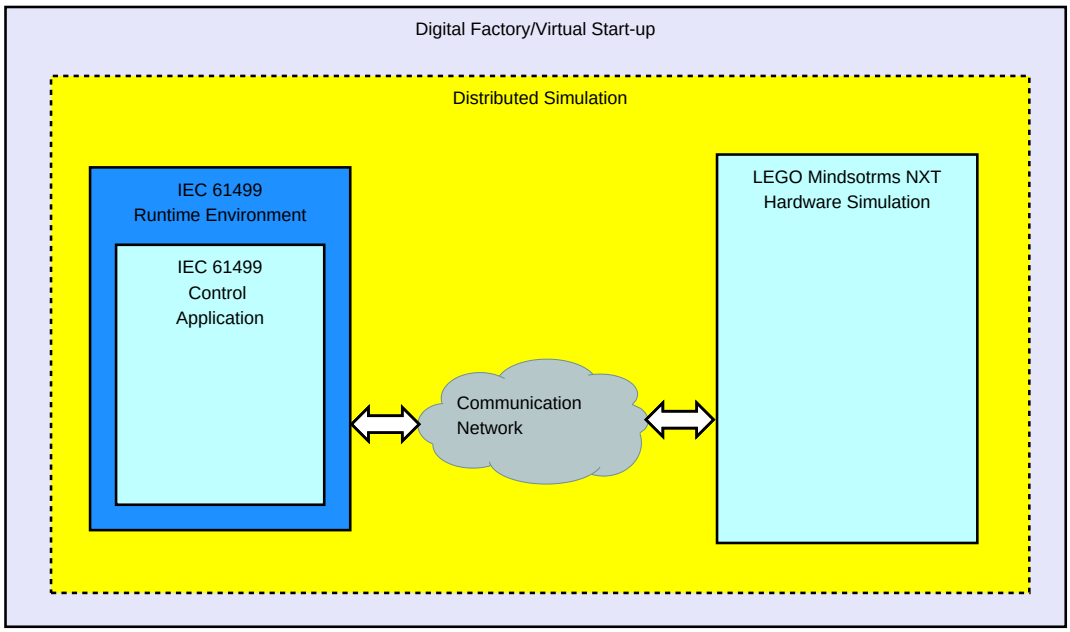


Figure 2.1: Connection of the topics in the *State of the Art*

We will start with the *Digital Factory*, since it is the framework, where the other topics like distributed simulation, IEC 61499, and the LEGO® Mindstorms NXT will fit into. *PLC program commissioning in a Digital Factory* describes the transition of the Digital Factory to the real factory and offers early testing and integration. Then we will deal with the individual parts of the simulation, starting with *IEC 61499*, which will be used to develop the control application in the simulation. This

application will be executed in an emulated environment, namely FORTE, which is a part of 4DIAC. For the other part of the simulation, which is the simulation of the *LEGO® Mindstorms NXT hardware*, an overview of the bricks used in this thesis will be given. The simulation will be developed using a *3D game engine*, so the features of such a game engine are presented as the next topic in this section. Last but not least, *distributed simulation*, *clock synchronization* and the *communication network* are important parts, which are needed to connect the different parts of the simulation.

## 2.1 Digital Factory

In this section the key features of a Digital Factory will be introduced. The underlying concept will be discussed, and the connection to our problem outlined, since it will form the basic framework for the implementation. Now we will start from the historical point of view, how and why the concept of the Digital Factories is being used.

The traditional approach to building factories was mainly a serial one. The basis for the design was the mechanical construction, where for the actors and processes were defined for the automation process. The plans were designed, and used for the hardware and software design. The real control application could earliest be tested at the construction site, when all the components were in place and wired. The quality of the software also used to be low until that step, since it could not be tested. There used to be a big separation between mechanical construction and automation. One goal of the Digital Factory is, to settle important trans-sectoral information as soon as possible during the design phase. With that information the time needed for the design and the quality of the planning process and construction results can be decreased. The functions of the sensors of the facility are being mapped into the simulation model. Therefore the quality of the system, concerning collisions and mechanical characteristics, is being increased. [24]

### 2.1.1 Basic Concept

The term *Digital Factory* can be split up into two main directions, out of which only the second is of interest for our purposes, and will be discussed in greater detail. According to [4], the two directions are :

- The design of new products with simulative methods, leading to whole simulation of assembly lines.

- The design, planning, and measurement of productivity of factories, based on virtual designs, reaching from machines to the connection of whole production cells.

At the moment, simulations of factories is achieved by creating basic models of the components, and to connect them to bigger units. Those models are usually created with powerful CAD design tools, and augmented with the necessary interface for interaction. What is missing in those models, is the mechanical part. That is, having spacial, dynamical and temporal behavior of the facility units, and of their interaction. [4]

The augmentation of the models results in an almost complete virtual model of the machine or the factory, with which the conflict-free interaction of the single parts can be testet. Those models need a lot of computing power, which is the limiting factor to the complexity of the models. Usually the closed design is used in those simulations, that is the separation of mechanic (Geometry, inertia, dynamics) and logic (logical constraints, user control, etc.). [4]

We can already gain a first impression of what the concept of the Digital Factory means for our problem: The Digital Factory can be used as a framework for our virtual environment, which will be split into a mechanic, and a logic simulation. Now we will further expand the concept of the Digital Factory.

According to [4], the requirements for a simulated plant are:

- Correct temporal behavior of all parts of the factory

- Correct display of all logical constraints

- Integration of all integrated means of transportation

- Display of all material flow

The complexity of the simulation of the factory should be adjustable, to be able to test the single units in a very detailed manner, but only to have the most important data when the whole factory is simulated. This should also reduce the requirements of computation power.

According to [4], the simulation can be split into two parts, in order to be able to take out a big factor of the complexity.

- Tier 1: Geometry, degrees of freedom, dynamics, temporal behavior, interfaces to sensors and actuators.
- Tier 2: Logic, Interfaces to sensors and actuators, states of the factory, interfaces to higher level systems.

The connection to our solution can easily be made. The two tiers described here, relate to the IEC 61499 Control Application (tier 2), and the LEGO® Mindstorms NXT Hardware Simulation(tier 1). This relation can also be seen in the overview Figure 2.1 at the beginning of this chapter.

According to [4], for the tier 2 there are several possibilities of realization:

- To be integrated into the model as an encapsulated part.
- Within Soft-PLCs, which are emulations of PLC devices, running on a PC.
- Direct realization within the real PLC devices, which will be used in the factory.

Regarding the cost and benefit of a Digital Factory, estimations of the leading suppliers of simulation systems speak of 10% to 24% cost reduction, using modern design and simulation methods, compared to traditional approaches [4].

A concept, which is designed to be used in industry also needs to be commercially attractive, so a benefit of 10% to 24% is a huge advantage over traditional approaches. The handling of the product and the product engineering in the Digital Factory will be the topic of the next section.

### 2.1.2 Product and Production Engineering

As [31] put it, the Digital Factory concept can also be seen as an enterprise and information strategy managing and collaborating processes of factories in global networks, as illustrated in Figure 2.2. It offers methods and software solutions for product and portfolio planning, digital product development, digital manufacturing, sales and support that deliver faster time-to-value, as shown in Figure 2.3. Collab-

orative solutions support people and processes involved in each major phase of the product and production phase [6].

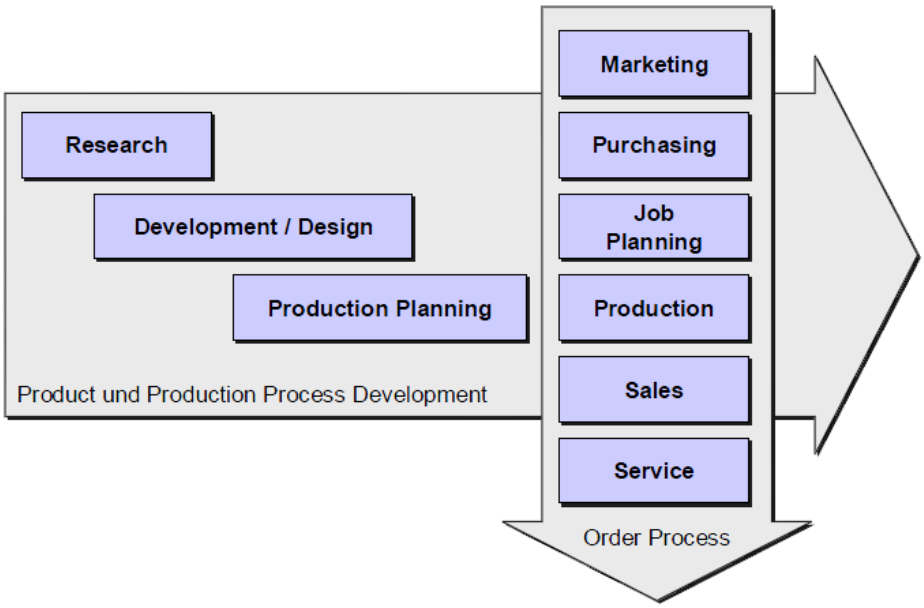


Figure 2.2: Digital Factory - Processes [31]

We can conclude that also for product planning, we can survey a benefit in the form of cost reduction, when we apply the Digital Factory concept. So we found good reasons why we should apply the concept to facility planning and product planning. In this thesis we will not consider product planning. Still it has been taken into account, to provide an all-embracing point of view.

## 2.2 PLC Program Commissioning in a Digital Factory

In the previous section the concept of the Digital Factory has been introduced. We have delineated the terms *tier 1* (simulation part) and *tier 2* (control part). For the tier 2, we defined three ways of realization: It can be integrated into the model as an encapsulated part, or within Soft-PLC, which are emulations of PLC devices, running on a PC, or direct realization within the real PLC devices, which will be used in the factory. Those variations are used in the concept of PLC program commissioning in a Digital Factory. Usually, first the PLC programs are executed within an emulation environment, and are transferred to the real PLC later on, where it still interacts with tier 1 (simulation part).

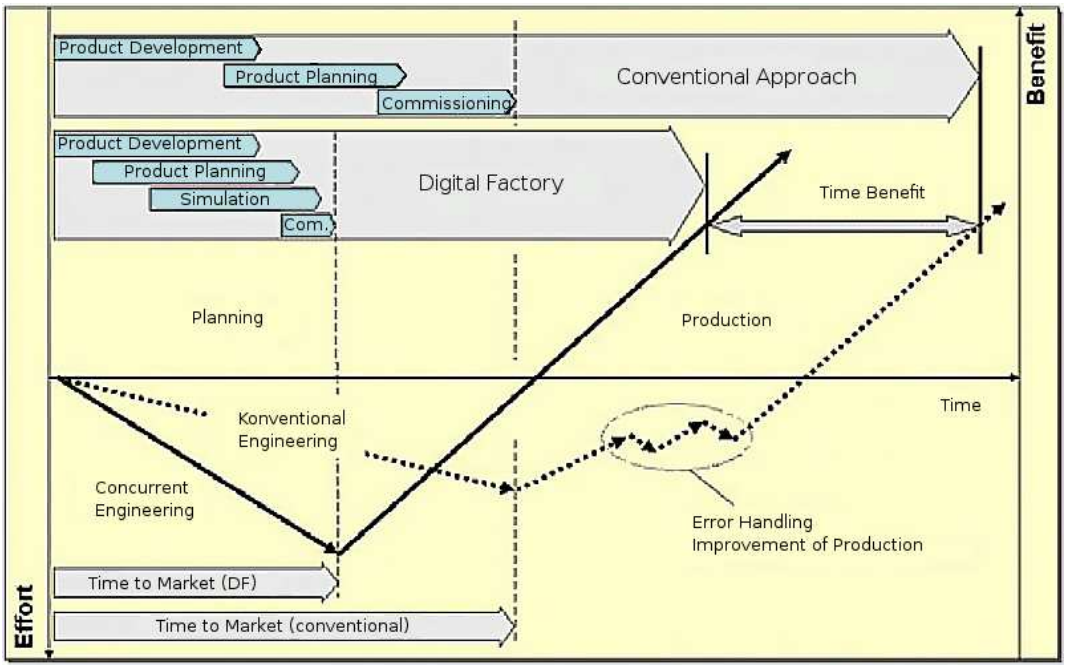


Figure 2.3: Digital Factory - Benefit and Effort [31]

We have come from the general concept of the simulation of both the control and the simulation part in the Digital Factory, to the point, where the control part is already executed on the real PLC but still interacts with the simulation. But would it also make sense to connect a Soft-PLC to the real factory? The answer should be no, since the whole benefit of the Digital Factory results from the simulation part, and early error elimination. It would not make sense to build the factory, just to connect it to the (Soft-)PLC, so the control application can be tested. In the whole system the real factory is the most expensive part to change, and the whole concept of the Digital Factory should eliminate the need of changes, once the factory is built.

The connection to our problem can be established here. In the context of this chapter, in our solution, the simulation part of the LEGO® sensors and actuators would stay the same, but tier 2, which is the IEC 61499 control application, would either be executed within an emulation of a PLC, or the real PLC itself. In our case, the control application will either be run on a PC, or on the LEGO® Mindstorms NXT Controller brick.

There are several terms for *PLC program commissioning in a Digital Factory*. Two of them are *Soft Commissioning®* and *Virtual Start-up*. Those two concepts will be presented in the next sections.

### 2.2.1 Soft Commissioning®

The basic idea of Soft Commissioning® (SoftCom) is to test industrial control software by connecting a controller, for example a PLC to a discrete event simulator (DES), which provides system reactions and sensor signals similar to the behavior of real hardware, for example an industrial manufacturing line [27].

Testing the behavior of a PLC, which controls a device being part of a more complex system, is usually done by connecting the controller to a stand-alone version of the device, a so called mock-up. This method of verifying and validating the controller's software is expensive, and test conditions are hard to reproduce. The tester also has to put up with the fact that such tests are incomplete since the interaction of this device with the other parts of the system is simply ignored. Therefore a large part of testing and debugging is still carried out on-site. Both approaches rise additional costs and might lead to the destruction of the device or even endanger human life when testing the controller's reaction to critical situations. Soft Commissioning® is a Hardware-In-the-Loop (HIL)-based approach to solve such problems. HIL means that the inputs and outputs of a controller are connected to a simulation (emulation) of the part to be controlled. This enables better reproduction of test conditions and even allows the tester to reproduce the interaction of the various parts of the complete system. [27]

### 2.2.2 Virtual Start-up

Virtual Start-up represents the transition from the Digital Factory to the real factory. For Virtual Start-up the facility is modeled digitally with all relevant automation properties and is run with the real PLC program. As a project milestone before the manufacturing of the mechanical components virtual start up verifies the planning and leads to a shortened, ensured ramp-up of the real facility. [24]

According to [24] during the Virtual Start-up the model of the facility is already controlled with the real control application, in the best case even with the real control devices. The control application can be generated automatically, or in a traditional way. The important thing is, that the model of the facility is accurate concerning the sensors, actuators and the behavior of the real factory.

In the Table 2.1 we can see the effects of the transition from the virtual to the real factory on the product, process and the resources.

| | *DigitalFactory* | ***Virtual Start-up*** | *Realfactory* |
|---|---|---|---|
| *Product* | Virtual Product | **Virtual Product** | Real Product |
| *Process* | Process Controlled | **Real Control** | Real Control |
| *Resource* | Virtual Resource | **Virtual Resource** | Real Resource |

Table 2.1: Virtual Start-up between the digital and the real factory [24].

## 2.3 IEC 61499

In this Chapter we will gain a short overview of the International Electrotechnical Commission (IEC) 61499 standard. If we take another look at the overview Figure 2.1 at the beginning of the chapter, we can see, that IEC 61499 is used to form the control application part of the Digital Factory. It will be executed in a runtime environment, which is a Soft-PLC on a PC or on the LEGO® Mindstorms NXT Controller brick. Also the execution environment will be presented in this section.

The IEC 61499 standard defines a modeling approach using Function Blocks. A Function Block is a functional unit of software that has its own data structure, which can be manipulated by one or more algorithms. It can be connected with others to form a network. A Function Block possesses the required characteristics of a software component defined as a unit of composition with contractually-specified interfaces and explicit context dependencies only. A software component can be deployed independently and is object to composition by third parties. The primary purpose of the standard is not as a programming methodology, but an architecture and model for distributed systems. IEC 61499 provides a set of models for describing distributed systems that have been programmed using Function Blocks. Thefore it is not designed directly to be used by programming tools, but provides terminology, models and concepts to allow the implementation of a Function Block oriented distributed control system to be described in an unambiguous and formal manner. [5, 26, 28, 29]

Since IEC 61499 is not a programming language, we also cannot say that the control application is developed *in* IEC 61499, but it can be developed using IEC 61499 compatible tools. Especially in the automation industry, the idea of Function Blocks is popular, since they reflect the idea of wiring. The next section will describe the basic concepts of IEC 61499:

### 2.3.1 System Architecture

There are various models introduced by IEC 61499, that together form the architecture for a Function Block oriented distributed systems:

**System model:** At the physical level, a distributed system consists of a set of devices interconnected by various networks to form a set of co-operating applications. An application will typically require the interoperation of software running in a number of devices. There can be a central processing device, controlling connected devices, or the system can be really distributed, where no central controlling device can be identified anymore. [26]

An application can exist on a single device or have functionality distributed over a number of devices. A distributed application will be designed as a network of connected Function Blocks. Communication services provided by each device ensure that Function Blocks that form part of an application maintain their data and event connections. According to [26] we can provide an overview of IEC 61499:

**Function Block model:** At the core of the standard is the Function Block model that underpins the whole IEC 61499 architecture. A Function Block is described as a functional unit of software that has its own data structure which can be manipulated by one or more algorithms. There are several types of Function Blocks:

**Basic Function Block:** The behavior of a basic Function Block is defined in terms of algorithms that are invoked in response to input events. As algorithms execute they trigger output events to signal that certain state changes have occurred within the block. The mapping of events on to algorithms is expressed using a special state transition notation called an Execution Control Chart (ECC).

**Composite Function Block:** The internal behavior of a Composite Function Block and subapplication types is defined by a network of Function Block instances. The definition therefore includes data and event connections that need to exist between the internal Function Block instances.

**Service Interface Function Block:** Service Interface (SI) Function Blocks provide an interface between the Function Block domain and external services, for example to communicate with Function Blocks in a remote device or to read the value of a hardware real-time clock. Because an

SI Function Block type is primarily concerned with data transactions, it is defined using time sequence diagrams. This form of diagram is more commonly used to define transactions across communication interfaces.

**Application model:** An IEC 61499 application is defined as a network of interconnected Function Blocks, linked by event and data flows. An application can be fragmented and distributed over multiple resources. With an application further decomposition is possible using subapplications. A subapplication has the external characteristics of a Function Block, but can contain networks of Function Blocks that can, themselves, be distributed over other resources. The application defines the relationships between events and data flows that are required between the various blocks. An application consists of Function Block instances and interconnection definitions which, in some cases, includes multiple instances of Function Blocks of particular block types.

**Device model:** A device is able to support one or more resources. A resource provides independent execution and control of networks of Function Blocks. The device has a process interface that provides the service that enable resources to exchange data with the input and output points on the physical device. There is also a communication interface that provides communication services for resources to exchange data via external networks with resources in remote devices.

**Resource model:** The resource provides facilities and services to support the execution of one or more Function Block application fragments. Function Blocks of a distributed system will be allocated to resources within interconnected devices. The resource provides interfaces to the communication systems and to external services and sub-systems that are closely connected to the device, such as the device I/O sub-system. The resource is therefore concerned with the mapping of data and event flows which pass between Function Blocks in the local resource to remote resource Function Blocks via the device communication interfaces.

**Distribution model:** There is an important differentiation between Function Blocks and applications. Both applications and subapplications can be distributed, that is, configured to run on several resources. A distributed application will consist of a network of Function Blocks with fragments of blocks running on designated resources. Because a device can support multiple resources, it is possible for a distributed application to run on a single device. In contrast, Function Blocks are assumed to be atomic and to only run in a single resource.

IEC 61499 offers a variety of models, which can be used for our purposes. Using integrated development platforms like 4DIAC-IDE [2] we can combine those models to whole systems. The framework for the control application and the control application itself will be developed using those models. Since we want to have a Digital Factory, we also need a Soft-PLC, which is a runtime environment for IEC 61499 applications. An overview of those execution environments will be given in the next section.

### 2.3.2 Execution Environments

There are several execution environments for IEC 61499 Function Blocks like Fuber [7], $\mu$Crons [21], FBRT [9] and FORTE [9]. Because of its open and extendable design, 4DIAC-RTE/FORTE is requested to be used in the thesis, and only this tool will be described more closely. The 4DIAC-RTE (FORTE) is a small portable implementation of an IEC 61499 runtime environment targeting small embedded control devices (16/32 Bit). It provides the execution of basic Function Blocks, composite Function Blocks, and service interface Function Blocks. The following two versions are supported: A PC based version tested on i386 (Cygwin and Linux) and PPC (Linux), and an embedded ARM7 based version.

## 2.4 LEGO® Mindstorms NXT Hardware

The LEGO® Mindstorms NXT hardware has a strong representation in our project. On the one side, the sensors and actuators are being simulated and on the other side, the NXT Control brick will serve as the target platform for the *PLC Program Commissioning in the Digital Factory*, which has already been discussed in this chapter.

The LEGO® Mindstorms NXT product package consists of sensors, actuators, and a controller, which is depicted in Figure 2.4. Those blocks can be used to build robots, but also to build automation arrangements. Those arrangements are no complete substitute to complex automation hardware, but key features are provided, and in some setups all required features can be realized. The benefit is that the sensors and actuators easily connect to the control brick. This control brick can be programmed using a variety of tools. Also there are projects to put a customized firmware into the control brick. That unlocks a variety of possibilities. One of those projects enables the control brick to execute IEC 61499 compatible code. In order to enable that

feature, the previously discussed *FORTE* will be run on the NXT controller. That way the same programs can be executed on the PC, and on the LEGO® Mindstorms NXT controller.

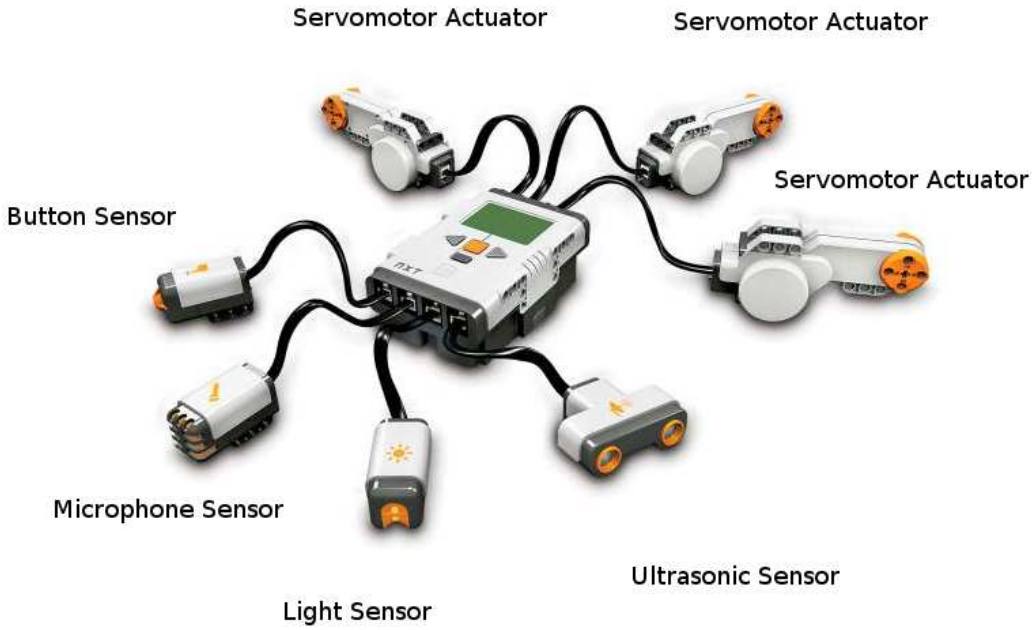


Figure 2.4: LEGO® Mindstorms NXT control brick, sensors, and actuators [17].

Another big benefit of LEGO® for our purposes is, that there are several sources for 3D models of the bricks available. This comes in very handy, using those models in a 3D game engine, and that way the focus can be on the development, and less time needs to be put into modeling. The package consists of the following list of sensors, actuators and control devices, which will be used in the simulation later on.

**Intelligent Brick:** The Intelligent Brick is the control device of the bundle. It has four ports for sensors, and three ports for actuators. The key features of the Brick are [17]:

- 32-bit ARM7 microprocessor (AT91SAM7S256), ATmega48 Co-processor
- Support for Bluetooth wireless communication
- 1 USB 2.0 port

- 4 input ports, 6-wire interface supporting both digital and analog interface
- 1 high speed port, IEC 61158 Type 4/EN 50170 compliant
- 3 output ports 6-wire interface supporting input from encoders
- 100x64 pixel LCD black & white graphical display
- Sound output channel with 8-bit resolution
- 4 button user interface
- 6-wire industry-standard connector, RJ12 right side adjustment
- Powered by 6 AA (1.5V) Batteries

The Controller brick/Intelligent brick can be seen in Figure 2.4. It is the brick in the middle, which the sensors and actuators are connected to. Those sensors and actuators will be presented now:

**Interactive Servo Motor:** The Servo Motor has a built-in rotation sensor that measures speed and distance, and reports back to the NXT Intelligent Brick. This allows for precise steps and complete motor control within one degree of accuracy. Several motors can be aligned to drive at the same speed.

**Touch Sensor:** The Touch Sensor reacts to touch and release. It can detect single or multiple button presses. The touch sensor is a passive sensor, and is sampled every 3ms.

**Light Sensor:** The Light Sensor can distinguish between light and dark. It can determine the light intensity in a room or the light intensity of different colors. The light sensor is a passive sensor, and is sampled every 3ms.

**Ultrasonic Sensor:** The Ultrasonic Sensor supports the Intelligent Brick with information about the distance of objects from the sensor. The ultrasonic sensor is a digital sensor, which means it uses the $I^2C$ communication, which needs an external micro-controller that handles the sampling of the physical environment.

## 2.5 3D Game Engines

The 3D game engine will play a crucial part in our solution. It will be used for the visualization of the simulation part of the Digital Factory. To gain a better overview, how it fits into the overall idea, as in the previous chapters, Figure 2.1 should be considered. In the game engine, the sensors and actuators of the LEGO® Mindstorms NXT package will be displayed, and animated. Some functions, like the computation of collisions will be done by the game engine. In this section the features of a game engine will be described, and the history of its development outlined.

A 3D engine is some software component, that provides rendering of the 3D scene in real-time, which means to load and display textures and sprites, display text, render special effects like fire, fog, and water. Furthermore it provides means to load, display, and animate 3D models. Most of the game engines also come with support of a sound system, physic system, I/O-system, network support, and data management.

From the history point of view, early computer games were developed from scratch. That meant redevelopment of the same structure and code for each computer game. With growing computation power, also the complexity of the games increased, and higher level programming languages were used. That resulted in complex applications, and better structuring of the code was needed. The focus changed from performance tweaking and shrinking the memory usage to better design, more features and more appealing graphics. An outline of the components of an interactive game is given in Figure 2.5. [14]

As we define it, a game engine includes all elements in Figure 2.5 that have no effect on actual content, that is, everything indicated by dashed lines plus an event loop. Further improvement in computation power allowed development of 3D games. One of the main techniques used here is scene management, which is used to indicate which parts of a scene are potentially visible to a viewpoint. With special graphic cards, this rendering becomes faster, since functions for computation are realized in hardware in those graphic cards, and the CPU does not have to compute everything. [14]

A game engine also needs to provide an application programming interface (API), which is the interface between the engine and the application, offering a variety of functionality to the programmer. That way developers can create content, without having to deal with low level functions like sound and graphics programming.

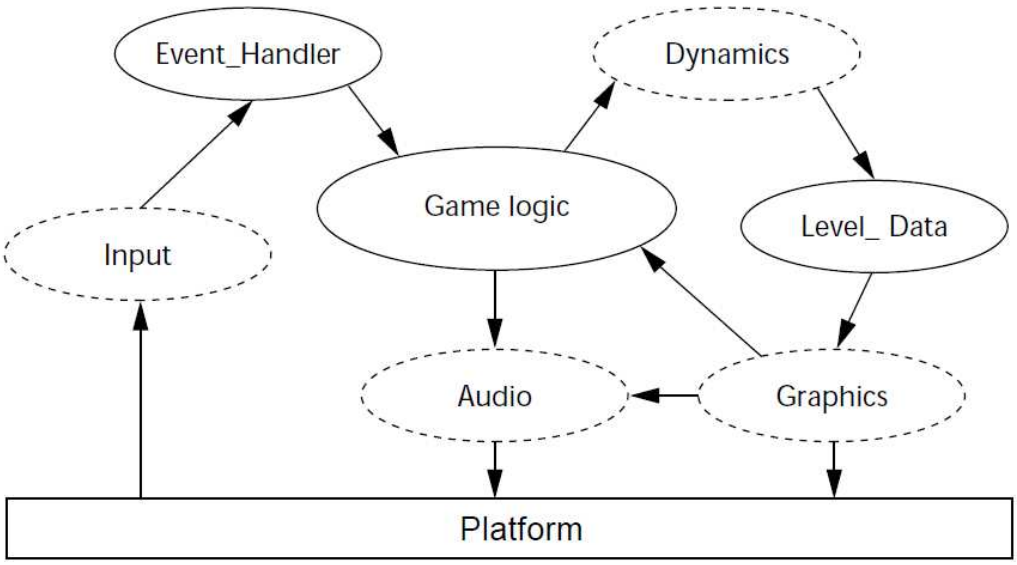


Figure 2.5: Game Engine Overview [14]

Lately some game engines provide very high level editors, where the games are develop rapidly, in a data-driven manner. The other remarkable development is, that game engines are more and more used for serious areas like visualization, training, medical and military simulations, which leads to the following distinction [32]:

**Game** a physical or mental contest, played according to specific rules, with the goal of amusing or rewarding the participant.

**Video Game** a mental contest, played with a computer according to certain rules for amusement, recreation, or winning a stake.

**Serious Game** a mental contest, played with a computer in accordance with specific rules that uses entertainment to advance government or corporate training, education, health, public policy, and strategic communication objectives.

The outcome of this thesis will belong to the area of the *Serious Games*. Since we have to deal with two simulation systems in our *Serious Games*, we have to examine *Distributed Simulation*, which is the topic of the next section.

## 2.6 Distributed Simulation

In factories the sensors, actuators, control devices, products, etc. are usually not integrated in one device. Therefore simulations within the Digital Factory have to deal with distributed devices, resulting in a distributed simulation. Also in the overview Figure 2.1 the position of the distributed simulation within the Digital Factory can be observed. We will start with a definition of distributed simulation:

> "*Distributed simulation is concerned with the execution of simulations on geographically distributed computers interconnected via a local area and/or wide area network.*" [25]

There are several reasons one might want to distribute the execution of a simulation across multiple computers. The most important of those reasons are listed here [25]:

**Reduced execution time:** By subdividing a large simulation computation into many subcomputations that can execute concurrently one can reduce the execution time by up to a factor equal to the number of processors that are used.

**Geographical distribution:** Executing the simulation program on a set of geographically distributed computers enables one to create virtual worlds with multiple participants that are physically located at different sites.

**Integrating simulators from different manufacturers:** If several simulations for different purposes already exist, it might be a good idea to connect those simulations to form a new one.

**Fault tolerance:** Another potential benefit of utilizing multiple processors is increased tolerance to failures. If one processor fails, it may be possible for other processors to continue the simulation provided critical elements do not reside on the failed processors.

For our purposes, *geographical distribution* is the main motivation to use distributed simulation. In factories the devices are usually spread out over the whole facility, so also the simulation needs to take that distribution into account. Also *Integrating simulators from different manufacturers* might be of interest, since existing simulations might have to be integrated into a Digital Factory. Now we will learn about the classification of distributed simulations. There are two classes of distributed simulations [25]:

**Analytic simulations** are typically used to quantitatively analyze the behavior of systems, for example to identify bottlenecks in a factory assembly line. These simulations usually run *as-fast-as-possible* meaning the goal is to complete the simulation execution as quickly as possible. They may run as *batch* programs without human intervention, or may include an animation to depict the operation of the system being modeled.

**Distributed virtual environments** are a newer class of simulation applications. Here, simulations are used to create virtual worlds into which humans can be embedded for training or entertainment.

Our solution will contain parts of both classes of distributed simulations. There will be a virtual environment, which offers an interface to the human, but also the behavior of the system can be analyzed. The system will not run *as-fast-as-possible*, but will reflect execution speed of the real devices.
Speaking of execution speed, also time plays a crucial part in distributed simulation. The simulated devices not only need their own track of time, but they also have their own temporal behavior. For example there is a difference in the latency, between a short direct cable connection, and a connection over a communication network, where the communication partners are separated by several network components, which all add to the delay of the message. This temporal behavior has to be considered in the distributed simulation. On the other hand, also network delay might be caused by the distribution of the simulation itself, and messages within the simulation are delayed by delivering them over a network. A solution to this problem is given provided by. [13]

A solution is presented in Figure 2.6. In the figure, Client is a component that requires input from the simulation. Servers are computational units performing dynamics simulation and other issues (e.g., Collision Detection). These Servers can communicate between each other directly either by using message passing or shared memory. These components are connected via a Manager collecting data from multiple sources and delivering data to multiple destinations. Components such as Motion Platform Controllers or Input/Output Controllers can be connected using the Manager. The most important function of the Manager is to break any tight couplings between the User Interface and the simulation environment.

The presented solution offers a useful approach to shorten delays in communication systems. In our solution, we still have the problem, to synchronize the two tiers of the simulation (Client and Servers in the term of [13]). Since both of them simulate the real world, they both have to take time into account, so they need to run synchronized. The basis of synchronized simulation, is clock synchronization,

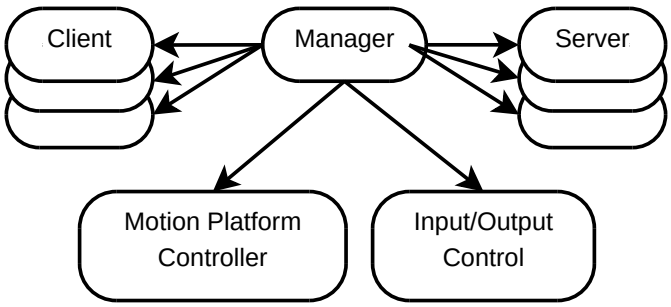


Figure 2.6: Distributed simulator structure [13]

which is the topic of the next chapter.

## 2.7 Clock Synchronization

Having distributed simulation, the different parts of the simulation might run on different computers. Those machines usually keep track of their own time. When communication depends on when a message has been received or sent, then the clocks of the different computers have to be synchronized. In order to synchronize the clocks, several algorithms exist. They will not all be discussed here, but the general problem will be described.

> *"Time synchronization in large distributed systems is the problem of giving all the physically disjoint elements of the system a common time scale on which to operate. This common time scale is usually achieved by periodically synchronizing the clock at each element to an ideal time source so that the local time seen by each element of the system is approximately the same. Time synchronization is often essential because it allows the entire system to cooperate and function as a cohesive group."* [3]

In a generic framework for the Digital Factory, where several devices might exist, all having their own track of time, synchronization is vital. In order to decide whether to react on a received message, or not, a device needs to have an idea how "old" the message is, and if it is outdated. This information is only available in synchronized systems with time-stamps in the messages. If a device reacts on an outdated message, unexpected, or even dangerous behavior might result. In our solution, time synchronization is not a problem, since the distributed simulation

is executed on one machine. Therefore only one "track of time" exists and no synchronization is needed. Still the problem of the temporal significance of the messages might arise.

## 2.8 Communication Network

The communication network is the connection of the two layers of the Digital Factory, and connects our distributed simulation. The requirements for the communication network vary from application to application. Horizontal (intra network) and vertical (inter network) homogenity in the networks is desired, so that the office, and the industrial networks can inter-operate.

In our solution the office network technology is of no interest, but still it has to be considered, since the office sector is more and more directly integrated into the production line. For the Digital Factory the requirements for the communication network might vary. There is a number of field bus technologies, which are used to cover different needs. If we want to design a generic framework of a Digital Factory, we need to support all of those different technologies. In order to avoid that problem a suggestion has been made [4]:

In order to avoid dealing with the different field bus systems during the phase of modeling of the facilities, a mapping of those field bus technologies to TCP/IP might be a good way to go.

In order to use TCP/IP, it also has to be supported by the network. Mainly two reasons have been found that can be considered for the use of Ethernet in industry [10]:

1. Existing field bus systems have reached a high complexity. In order to cover the future performance needs, a lot of effort has to be put into hardware and software development, since already today field bus systems have reached their limits of performance. To develop the new generation of control devices, as many synergies as possible have to be reached with the office networks, and the hardware and software has to be as standardized as possible.

2. Specialized tasks are covered by specialized networks, resulting in a lot of different networks. The following list shows a possible scenario how network technologies could be used in a factory:

a) Automation: Profibus

b) Vertical Communication: Ethernet

c) Image processing: Ethernet

d) Drive control: Sercos

e) Safety: SafetybusP

The mean to reach convergence here, is a universally usable network. For our purposes the desire to use TCP/IP and Ethernet in industry is very welcome, since our simulation will be executed on an ordinary PC, which usually supports those technologies "out of the box".

## 2.9 Summary

In this chapter we have introduced the fields of interest, which are important for this thesis. We have connected the state of the art with our problem, and got a first impression of the requirements and the significance of the topics for our solution. The Digital Factory was described, and the two-tier model presented, which consists of the simulation part (tier 1) and the control part (tier 2). Those two tiers will be used in our solution. Soft Commissioning® can be a part of the Digital Factory concept, as well as Virtual Start-up. Both describe the transition from the virtual to the real factory, and the combination of control applications which are executed in real PLC with the simulated facility.

In those control applications we will use IEC 61499 compatible code, so a definition of the IEC 61499 standard was provided, and the concepts which will be used in the solution outlined. The control applications will be run on an emulated PLC hardware. The tool of choice for the emulation is FORTE, which can also be executed on a LEGO® Mindstorms Controller. The LEGO® hardware, which will be used in our experiments was described.

The other part of the simulation, dealing with physical attributes, degrees of freedom and animated models, will be designed using a 3D game engine. An overview of the key features of a game engine was given. Last but not least, the communication network, connecting the two parts of the simulation, and clock/execution

synchronization were introduced. How those different topics are brought together will be described in the next chapter, the concept.

# 3 Concept

In the last chapter we discussed the fields of interest, which are relevant for our solution. We also got a first impression of some requirements, when we surveyed the *State of the Art*. Now we will pick up those requirements in order to expand and discuss them, so that we get a full list of requirements and can define the framework for the desired system. We will start with analyzing the system, which we want to develop. The results of the analysis will be used to design the system in the next step.

## 3.1 System Analysis

During the system analysis we will define the current state, and the target state. With current state we mean the state of the environment without the system. The target state will define the features of the system. Those two states will be compared to find out the requirements for the system.

### 3.1.1 Current State

Hardly any system can be designed, without taking its environment into account. Also for our system there are several conditions and presettings, which have an influence on it. We will define the framework into which our system will have to fit. Let us start with an overview of the current state of the environment, which is depicted in the Figure 3.1. We can see two systems, which we have to take into account when we design our framework:

**LEGO® Mindstorms NXT:** It is currently used in other projects at the University. One of them extends the controller so that FORTE can be executed on it. Therefore the controller is also compatible to IEC 61499, and control applications from the automation sector can be executed on it. Since the LEGO®

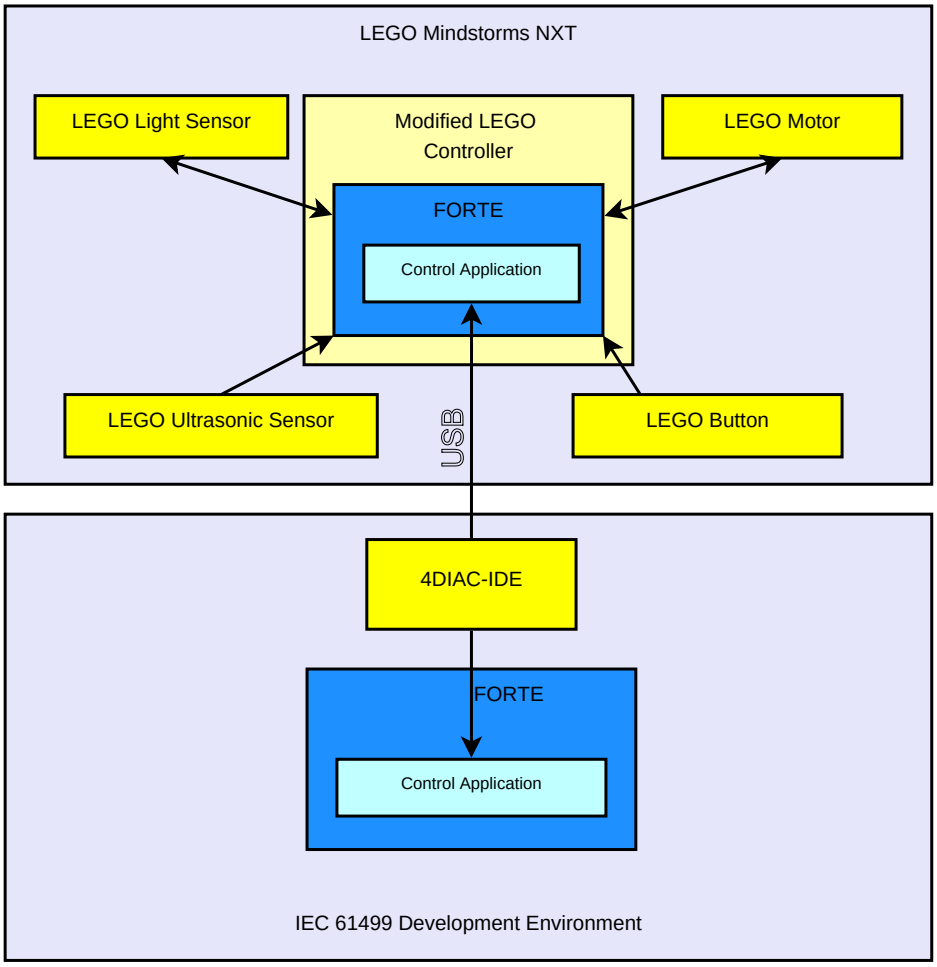


Figure 3.1: Current state of the environment

hardware is being used in other projects, it was one of the requirements from the *Task Description*, that it should be supported for the teaching purpose.

**IEC 61499 Development Environment** The other prerequitisite we have, is the support of the IEC 61499 Development Environment, which represents the currently used development platform for control applications at the institute. To reach maximum homogeneity in the usage of tools, 4DIAC will be used to create control applications, and FORTE will be used to emulate a PLC. Since the control application on a soft PLC lacks sensors and actuators to control, students who use a soft PLC can not control anything with their applications. To provide that missing part, a simulation framework should be designed, which offers controllable sensors and actuators.

We can summarize, that there are two environments, which will be integrated into our solution. On the one hand there is the LEGO® hardware, which is extended with FORTE, and on the other hand there is the IEC 61499 development environment, which will be used to develop the control applications.

### 3.1.2 Target State

Now we will come to the target state, which consists of the current state, extended by the desired system. It is depicted in Figure 3.2, where we can still see the *LEGO® Mindstorms NXT*, and the *IEC 61499 Development Environment*. What is new is the *Simulation Framework* in the middle. This will be the heart of our project, and will play the role of the simulation part of the Digital Factory.

As we can see, the simulation framework consists of the simulated sensors and actuators, which interact with the simulation experiment. The analogy to the LEGO® Mindstorms NXT system is quite obvious. Instead of the real sensors and actuators we have the simulated sensors and actuators. Here we can get one of the most important requirements: *The simulated sensors and actuators need to have the same behavior as the real sensors and actuators.*

The next thing that we can observe is the connection between the simulated sensors and actuators and the simulation experiment. This connection refers to the cable connection in the real LEGO® hardware. Therefore we need to *simulate the cable connection between the periphery and the controller*. We can also observe the simulation experiment in the simulation framework. This means, that different simulation experiments could be created, which access the same simulation framework. Therefore we need to *provide an interface to the functionality of the simulation framework*.

What we can also see is the connection between the simulation framework and both the LEGO® hardware and the IEC 61499 development environment. Since we want to simulate a factory, and in a real factory usually the periphery and the controller are distributed, we want the same design here. So the control application will not be a part of the simulation framework, but it also has to *offer an interface to the control application*. In a real factory, a field bus would connect the individual devices, so also in our solution we will use a communication network.

When the different parts of the simulation are executed on the same machine, then clock synchronization is not necessary, since only one track of time exists. But

still, since we simulate the LEGO® hardware, and the control hardware in different simulations, *synchronization of the execution* is needed.

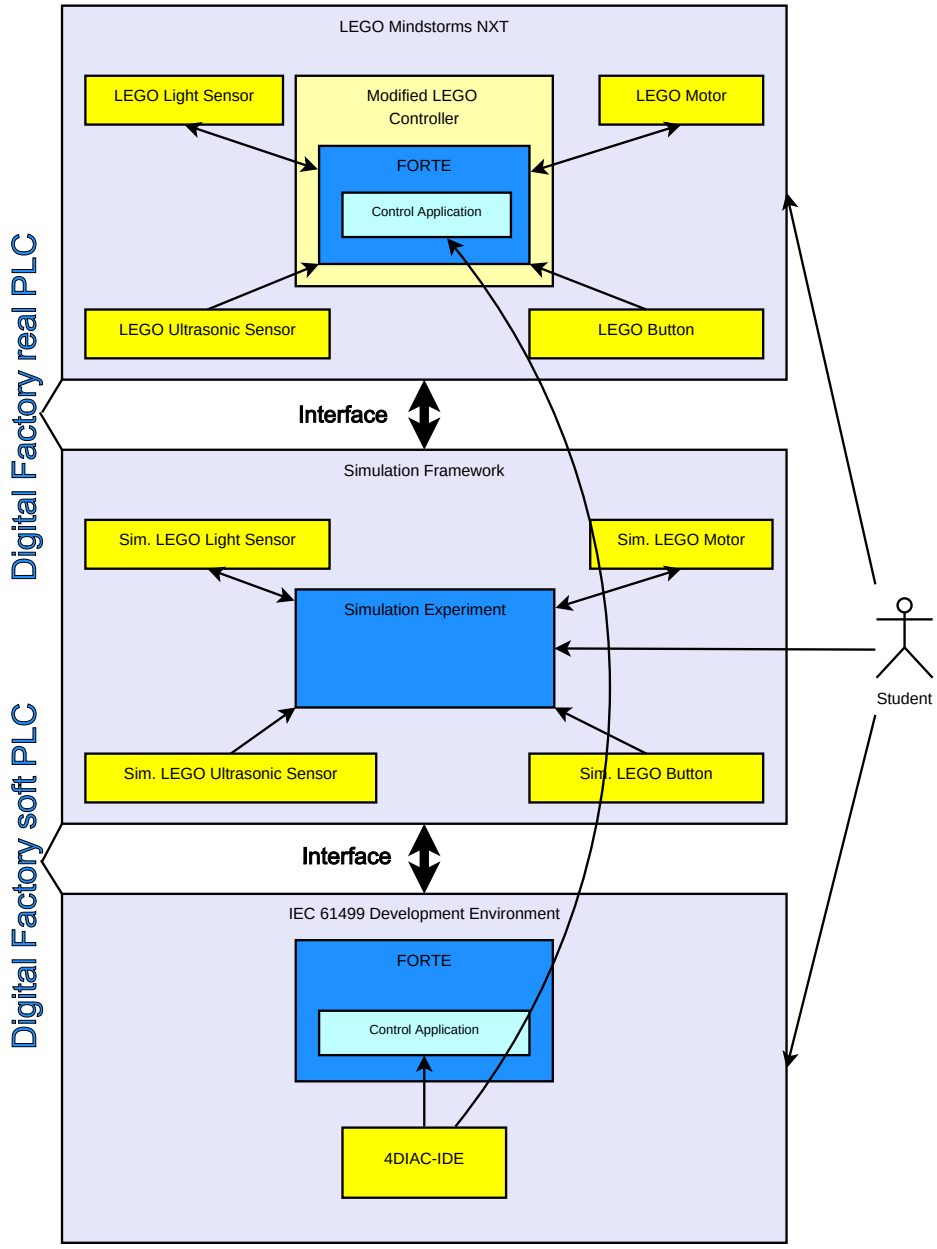


Figure 3.2: Target state with the target system

Another feature, which we want is the concept of Soft-Commissioning®. Therefore we need a soft PLC and a real PLC. Those roles will be assigned to FORTE on an

ordinary PC, for the soft PLC part, and the LEGO® controller with FORTE for the "real PLC". With this setup we can make *a transition from the simulated to the real PLC, both connected to the simulation framework.*

For the teaching purpose, we should have a look at the "Student" in Figure 3.2. The "Student" can access the IEC 61499 Development Environment to create control applications. Those control applications can be executed in the LEGO® controller, or in FORTE on a PC. Those features were already available before. Now, *the "Student" can also create a simulation experiment*, which could be a facility consisting of LEGO® bricks. Since the simulation framework also offers the connection to the control applications, *the "Student" can connect a control application to the simulation experiment*, and have a whole control loop according to the concept of the Digital Factory.

Analyzing the current state, and comparing it to the target state, we have found several requirements, which are summarized, in order to provide an overview in a compact way:

- Same behavior of the simulated and the real sensors and actuators.
- Simulate the cable connection between the periphery and the controller.
- Interface to the functionality of the simulation framework.
- Interface to the control application.
- Synchronization of the simulation and control applications.
- Transition from the simulated to the real PLC.
- The "Student" can create a simulation experiment.
- The "Student" can connect a control application to the simulation experiment.

Based on the list of requirements, we can now create a Use-Case diagram. First the diagram will be presented, and an explanation provided, then the Use-Cases will be described in structured text.

### 3.1.3 Use-Case Diagram

In the last section we have compared the current state with the target state. The differences have been listed, which gives us a rough idea of the requirements. Now we will categorize and extend those requirements and develop a Use-Case diagram. We will start with the textual description of the diagrams. The individual Use-Cases will be discussed in detail later in this section.

**Student Interface:** The relation between the Student and the Interface of the Simulation and the Control application can be seen in Figure 3.3. The Use-Cases represent the functions, that the interfaces can be accessed by an application, which was implemented by the Student, or being used in a user interface. The two interfaces are further explained in the next two diagrams.

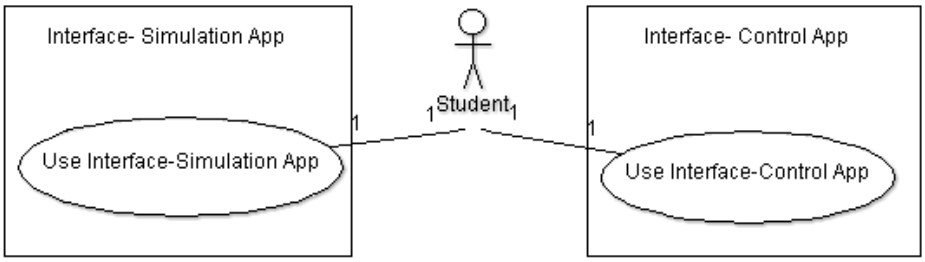


Figure 3.3: Student-Interface connection

**IEC 61499:** The diagram depicted in Figure 3.4 shows the IEC 61499 part of the framework, which offers the interface to develop control applications. This is in alignment with the control part of the Digital Factory (tier two). The actor in this diagram is the control application, which can be created by students. This means, that the control application uses the offered interface.

The actor can primarily access the Function Blocks of the sensors and actuators. The actuators only consist of the servo motor, so the Use Case *Access Motor FB* reflects that behavior. Accessing the motor FB includes setting the speed (and direction) the motor should have. This value is being sent through the Communication Network to the simulation part of the framework. Accessing the Motor FB also results in getting the Motor Ticks and the Client Status as outputs of the FB. Those values are being requested from the simulation part by sending the new value for the motor. The values- Motor Ticks and

the Client Status, which were received can be used by the Control Application again.

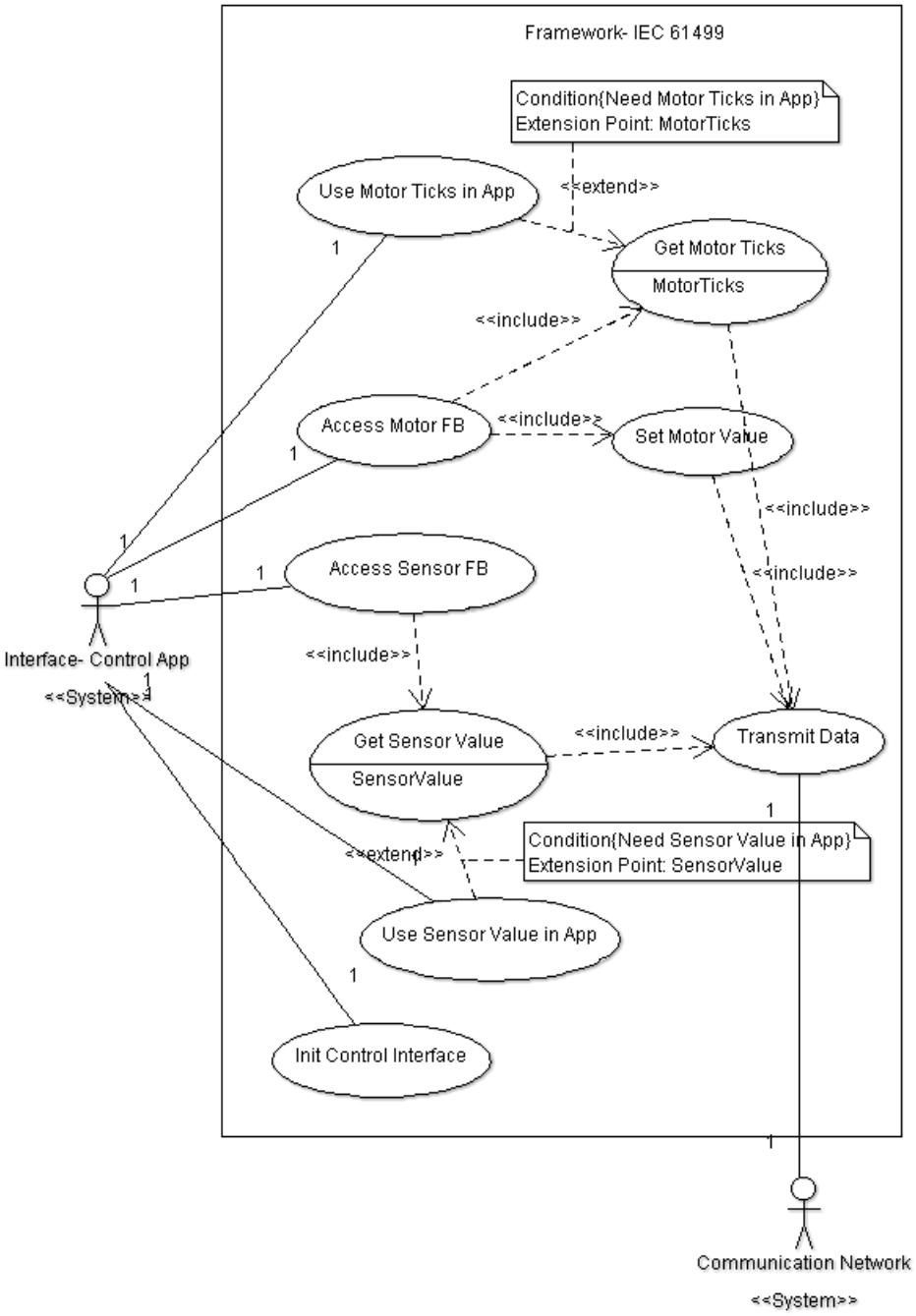


Figure 3.4: IEC1499 part of the framework

Almost the same principle as for the motor, works for the Sensors. The differences are, that there is more than one sensor (Button, Ultrasonic Sensor, Light Sensor), and that no value can be set. When accessing the Sensor FB, a signal is being sent internally, and answered by the current value of the sensor. Also the status of the client can be accessed and used in the application.

The last Use Case, which is accessible to the Control Application is the Init Control Interface. This includes the initialization of the communication connection.

**Communication Network:** Figure 3.5 shows two parts of the framework: the Communication Network, and the Simulation part. To start with the Communication Network, there is only one Use Case. This requires reliable data transmission. Since it is not clear, which communication protocol will be used, and might even vary, the only requirement is, that it needs to be reliable. That means that no packages will be lost, and no package may overtake another.

**Simulation:** The last part of the framework is the simulation part, which is also shown in Figure 3.5. If we start at the actor, we see that the actor is a «System», which is the interface that can be accessed by the simulation application. One of the Use Cases accessible to the SimulationApp is the Initialize Simulation Application. This includes the initialization of the communication connection. The simulation application can access simulation values, like the data, which is being transmitted, and set simulation options, like which models should be displayed in which way.

Now that we have an overview of the diagram, we can describe the actors and the Use-Cases themselves to complete the Use Case diagram. We will start with a list of the actors:

**Student** The Student can create own simulation and control applications, using the provided interfaces.

**Interface: Simulation App «System»** Using this interface the Student can access the functions of the “Framework- Simulation”.

**Interface: Control App «System»** Using this interface the Student can access the functions of the “Framework- IEC 61499”.

**Framework: IEC 61499 «System»** Representation of the System “Framework- IEC 61499”.

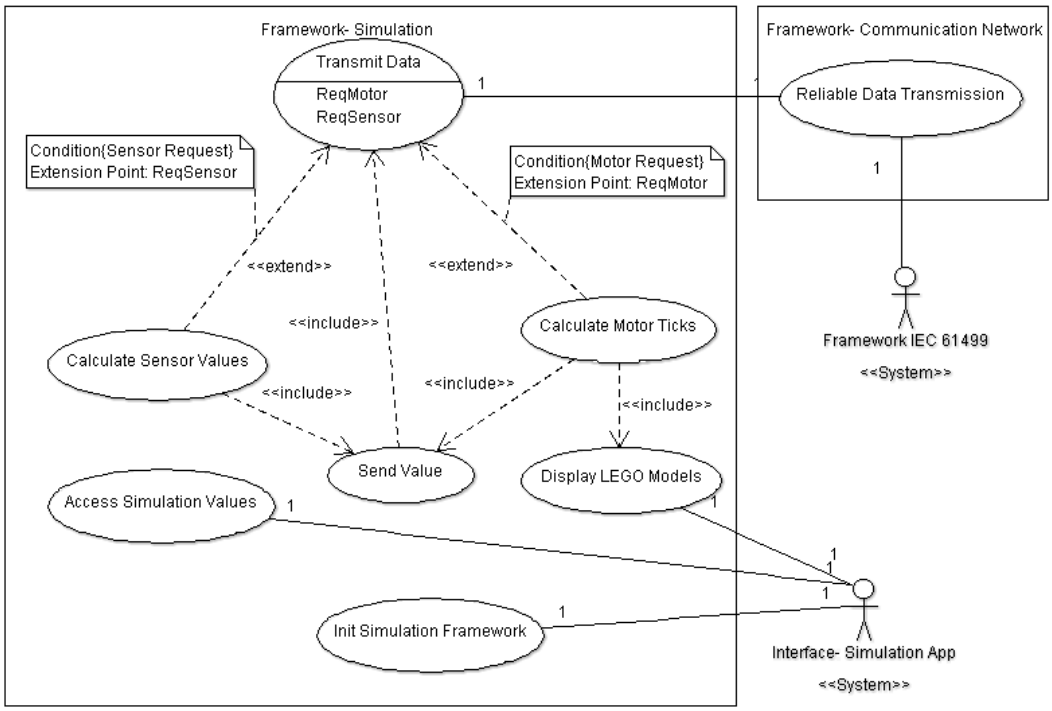


Figure 3.5: Communication network of the framework

**Framework: Communication Network «System»** Representation of the System "Framework- Communication Network".

We will continue with the description of the Use Cases:

**Use Case Interface- Simulation App:** Use Interface: Simulation App
**Description** The interface to the Simulation Framework, which is provided for the student is being accessed.
**Actors** Student
**Precondition** -
**Trigger** The Student accesses the Interface to the Simulation Framework.
**Basic Flow**

1. The Student accesses a function of the Simulation Framework.

2. The function is being executed.

**Use Case Interface- Control App:** Use Interface: Control App
**Description** The interface to the IEC 61499 Framework, which is provided

for the student is being accessed.
**Actors** Student
**Precondition** -
**Trigger** The Student accesses the Interface to the IEC 61499 Framework.
**Basic Flow**

1. The Student accesses a function of the IEC 61499 Framework.

2. The function is being executed.

**Use Case Framework IEC 61499:** Access Motor FB
**Description** The motor FB provides the functionality to set and request values of the simulated LEGO servomotor.
**Actors** Control Application
**Precondition** The connection to the simulation application must be established.
**Trigger** The user wants to set the speed and direction of the motor, read the motor ticks, or display the server status.
**Basic Flow**

1. The speed and direction must be provided as parameters.

2. The values are being sent to the simulation part.

3. The motor ticks are returned, and can be accessed.

**Use Case Framework IEC 61499:** Set Motor Value
**Description** The desired value of the motor is being sent to the simulation application.
**Actors** -
**Precondition** The motor FB has been accessed.
**Trigger** The motor FB has been accessed.
**Basic Flow**

1. The value for the motor is being sent to the simulation application.

**Use Case Framework IEC 61499:** Get Motor Ticks
**Description** The motor ticks are being requested from the simulation application.
**Actors** -
**Precondition** The motor FB is being accessed, and the Motor Value has been

set.
**Trigger** The motor ticks are sent by the simulation part.
**Basic Flow**

1. The motor ticks are received, and can be accessed.

**Use Case Framework IEC 61499:** Use Motor Ticks in App
**Description** The motor ticks can be accessed in the control application.
**Actors** Control Application
**Precondition** Motor ticks have been received from the simulation application.
**Trigger** The control application wants to use the motor ticks.
**Basic Flow**

1. The motor ticks are being used in the control application.

**Use Case Framework IEC 61499:** Access Sensor FB
**Description** The Sensor FB provides the functionality to set and request the values of the simulated LEGO sensors.
**Actors** Control Application
**Precondition** The connection to the simulation application must be established.
**Trigger** The user wants to request a value from a sensor, set a value, or display the server status.
**Basic Flow**

1. The value of a sensor is being requested from the simulation application.

2. The value is received and can be accessed.

**Use Case Framework IEC 61499:** Get Sensor Value
**Description** The sensor value is being requested from the simulation application.
**Actors** -
**Precondition** The sensor FB has been accessed.
**Trigger** The Sensor FB has been accessed.
**Basic Flow**

1. The sensor value is being requested from the simulation application.

2. The sensor value is received.

3. The sensor value and can be accessed.

**Use Case Framework IEC 61499:** Use Sensor Value in App
**Description** The sensor value is being used by the control application.
**Actors** Control Application
**Precondition** The sensor value has been requested.
**Trigger** The sensor value is being used in the control application.
**Basic Flow**

1. The sensor value is being used in the control application.

**Use Case Framework IEC 61499:** Init Control Interface
**Description** The control application is being initialized.
**Actors** Control Application
**Precondition** -
**Trigger** The control application requests an initialization of the interface.
**Basic Flow**

1. The communication connection to the simulation application is being initialized.

**Use Case Framework IEC 61499:** Transmit Data
**Description** Data is being sent to the simulation application.
**Actors** Communication Network
**Precondition** The control interface has been initialized.
**Trigger** Get Motor Ticks or Set Motor Value or Get Sensor Value or Init Control Interface want to send or receive data.
**Basic Flow**

1. Data is being exchanged with the communication network.

**Use Case Framework Communication Network:** Reliable Data Transmission
**Description** Data is being transmitted over the communication network in a reliable way.
**Actors** Framework IEC 61499, Framework Simulation
**Precondition** Initialized simulation framework, Initialized IEC 61499 framework.
**Trigger** Request to send data from the Framework IEC 61499 or the Framework Simulation.
**Basic Flow**

1. Request to send data.

2. Data is being transmitted.

**Use Case Framework- Simulation:** Display LEGO® Models
**Description** Display and animate the LEGO® Models.
**Actors** Simulation Application
**Precondition** -
**Trigger** Request to display the LEGO® Models
**Basic Flow**

1. Calculate the animations for the LEGO® models.

2. Display the LEGO® models.

**Use Case Framework- Simulation:** Calculate Motor Ticks
**Description** Calculate the Ticks of the motor.
**Actors** -
**Precondition** The values for the speed and direction of the motor have been received.
**Trigger** Request the ticks of the motor.
**Basic Flow**

1. Receive request for the ticks of the motor.

2. Calculate ticks.

3. Send the ticks to the control application and provide the values for the display of the LEGO® models.

**Use Case Framework- Simulation:** Calculate Sensor Values
**Description** Calculate the actual values of the simulated sensors.
**Actors** -
**Precondition** Get a request to calculate the sensor values.
**Trigger** Get a request to calculate the sensor values.
**Basic Flow**

1. Receive the request to calculate the sensor values.

2. Calculate the sensor values.

3. Send the sensor values.

**Use Case Framework- Simulation:** Send Value
**Description** Send a value to the control application.
**Actors** -
**Precondition** The framework has been initialized
**Trigger** Calculate Sensor Values or Calculate Motor Ticks
**Basic Flow**

1. Send the value to the control application.

**Use Case Framework- Simulation:** Init Simulation Framework
**Description** The connection to the control framework is being established.
**Actors** Simulation Application
**Precondition** -
**Trigger** Request to initialize simulation framework.
**Basic Flow**

1. Establish connection to the IEC 61499 framework.

**Use Case Framework- Simulation:** Transmit Data
**Description** Data is transmitted to the IEC 61499 framework, using the reliable data transmission.
**Actors** -
**Precondition** Simulation Framework has been initialized.
**Trigger** Receive data from the communication network, or send value request.
**Basic Flow**

1. Transmit the data over the communication network.

### 3.1.4 Identified Requirements

In the last section we have designed a Use-Case diagram. Several requirements can be obtained from it.We will categorize and extend those requirements and make a distinction between need, nice, and not to have.

### Need to have

- The two tier concept of the Digital Factory should be supported (a control part and a simulation part, which together form a control loop, connected by a communication network).
- The control part consists of an IEC 61499 framework, which has to be executable in FORTE.
- An interface has to be provided, so that IEC 61499 control applications can interact with the simulation framework.
- The simulation part consists of a simulation framework, which simulates the LEGO® hardware (servo motor, light sensor, ultrasonic sensor and the button).
- This simulated hardware can be controlled by the control application from the control part.
- The simulation will result in a serious game, using a 3D game engine, where the LEGO® hardware 3D models are displayed, animated, and can interact.
- Visual feedback will be given, and a user interface is provided, so that parameters of the simulation can be set.
- Within that framework experimental setups should be possible, where facilities can be simulated.
- The LEGO® hardware needs to be simulated in a physically correct way.
- The behavior of the simulated models needs to be the same as of the real hardware.
- Students need to be able to create a control application, which can access the simulation part, and control the simulated devices.
- The simulation and control applications need to run synchronized, and in real-time.
- The principle of Soft Commissioning® will be supported.

#### Nice to have

- An editor should be provided, so that the simulation experiments can be created in a "what you see is what you get" manner.
- Calculation of the physical interaction of the models using a physic engine.
- Support of more than one technology for the communication network.

#### Not to have

- Creating a framework, which supports the creation of a complete Digital Factory, due to its complexity (e.g., product engineering, assembly lines).

## 3.2 Overall System Design

In the last section we defined the requirements, which have to be fulfilled by the system. We also defined the Use-Cases, which provides a good overview of the necessary interfaces for the interaction with the environment. Now we will continue the system design, using a Class-Diagram. The class diagram is depicted in Figure 3.6. The two packages used are *Simulation*, which refers to the simulation part of the Digital Factory, and *IEC 61499*, which represents the control part of the Digital Factory. We will now provide a class description to the diagram to further clarify the classes and their interaction.

**Simulation-GameEngine:** The GameEngine class provides the functionality of the 3D game engine, which will be used and keeps track of the interaction between the models. It encapsulates the functions of the 3D game engine in order to have a clean interface. The interface provides all the necessary functions of the 3D game engine and internally calculates the interaction of the models.

**Simulation-UltrasonicSensor:** The UltrasonicSensor class represents the LEGO® ultrasonic sensor. It provides the function to read the value of the sensor. The value is updated by the GameEngine, and can be sent over the Communication class.

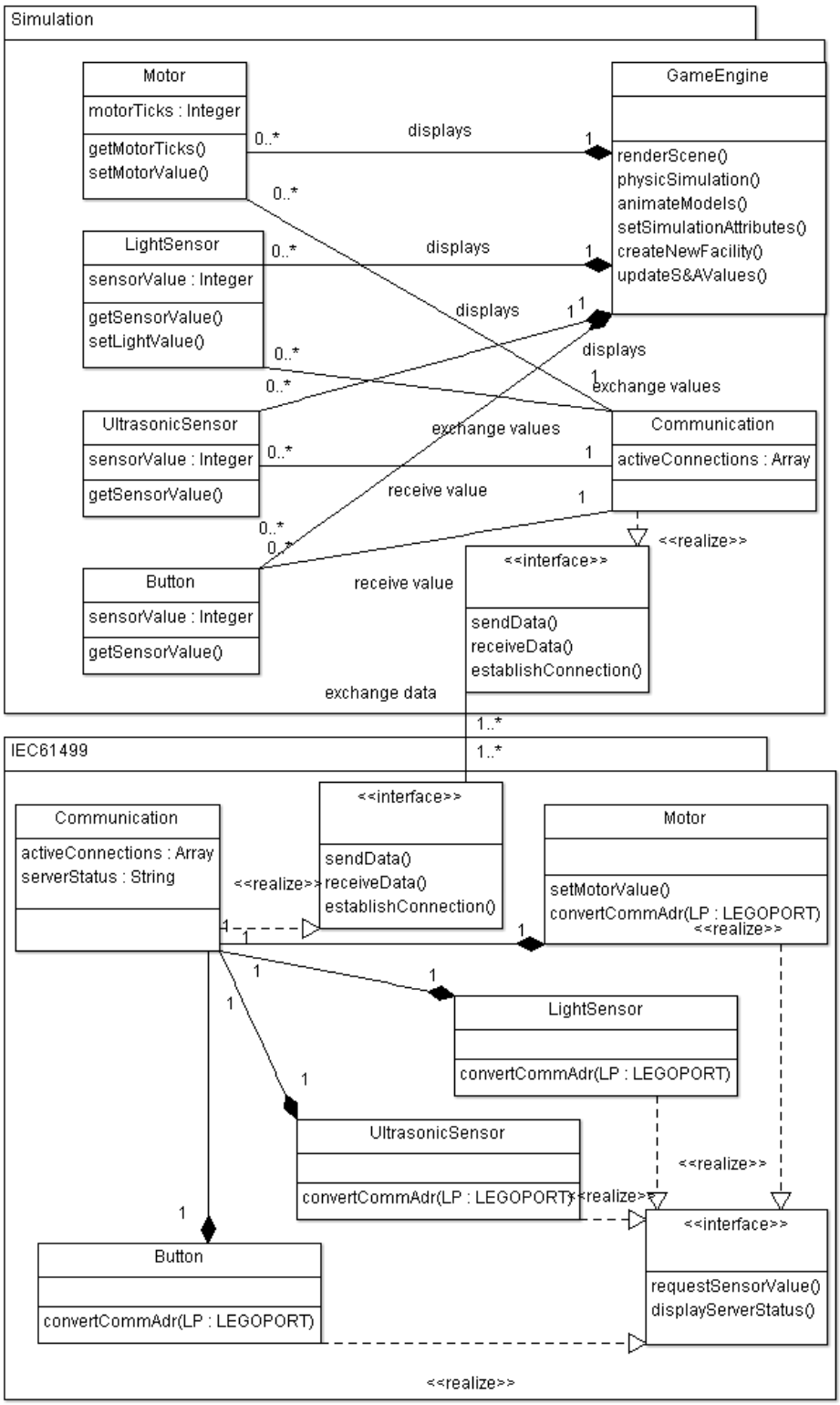


Figure 3.6: Class diagram of our system

**Simulation-Motor:** The Motor class is the representation of the LEGO® motor. It provides the functions to set the values (speed and direction) of the Motor model, and to read the MotorTicks. When the values are set, the Game Engine can calculate the animation and the physics according to the values, and update the MotorTicks, which can be sent by the Communication class.

**Simulation-LightSensor:** The LightSensor class represents the LEGO® light sensor. It provides the function to read the value of the sensor, and to turn on and off the integrated light bulb. The value is updated by the GameEngine, and can be sent over the Communication class.

**Simulation-Button:** The Button class represents the LEGO® button. It provides the function to read the value of the sensor. The value is updated by the Game Engine, and can be sent over the Communication class.

**Simulation-Communication:** The Communication class represents the communication protocol, and its handling (e.g., TCP/IP). The Communication class is necessary, in order to transmit data between the Simulation and the IEC 61499 packages. The data is transmitted over a network, or shared memory, in a reliable way.

**IEC 61499-Motor:** The Motor class represents an interface for the students, and emulates the interface from the LEGO® Controller to the sensors and actuators. The interface is used by the students to create their own control applications. The functions of the real interface between the LEGO® Controller to the sensors and actuators is being emulated and the parameters are internally converted to the need of the data transmission. (e.g., Motor Port "A" to TCP/IP address.

**IEC 61499-LightSensor:** The LightSensor class represents an interface for the students, and emulates the interface from the LEGO® Controller to the sensors and actuators. The interface is used by the students to create their own control applications. The functions of the real interface between the LEGO® Controller to the sensors and actuators is being emulated and the parameters are internally converted to the need of the data transmission (e.g., LightSensor on LEGO® Port "1" to TCP/IP address).

**IEC 61499-UltrasonicSensor:** The UltrasonicSensor class represents an interface for the students, and emulates the interface from the LEGO® Controller to the sensors and actuators. The interface is used by the students to create their own control applications. The functions of the real interface between the LEGO® Controller to the sensors and actuators is being emulated and the

parameters are internally converted to the need of the data transmission (e.g., UltrasonicSensor on LEGO® Port "2" to TCP/IP address).

**IEC 61499-Button:** The Button class represents an interface for the students, and emulates the interface from the LEGO® Controller to the sensors and actuators. The interface is used by the students to create their own control applications. The functions of the real interface between the LEGO® Controller to the sensors and actuators is being emulated and the parameters are internally converted to the need of the data transmission (e.g., Button on LEGO® Port "3" to TCP/IP address).

**IEC 61499-Communication:** The Communication class is a part of the IEC 61499 Motor, LightSensor, UltrasonicSensor, and Button. It provides the communication function for those classes. The Communication class is necessary, in order to transmit data between the IEC 61499 and the Simulation packages. The data is transmitted over a network, or shared memory, in a reliable way.

## 3.3 Summary

During this chapter the required system was analyzed and designed, which led to a definition of our system. We started with examining the current state, which is the environment without our system, then we added our system, and got the target state. Comparing those two states we gained some requirements. A Use-Case diagram further supported our search for requirements. Finally the requirements are categorized into "need to have", "nice to have", and "not to have".

From the list of requirements we learned what is needed to build the system. The question, how the system could be realized, was answered in the *System Design*. A class diagram was created, where the classes, and the interaction amongst the classes were defined. The class diagram completes the system design, and we can go on to the implementation.

# 4 Implementation

In the last chapter we specified the framework of the Digital Factory by using UML diagrams for analysis and design. With those diagrams the framework can be implemented now. We will start with the selection of the tools which are needed to develop the framework. That includes the IDE to be used, from which source we can gather the needed 3D models, which 3D engine we will use, and the selection of the communication technology. Then we will discuss how the control part, and the simulation part of the Digital Factory are implemented.

## 4.1 Development Environment

In this section the necessary design decision will be made. Especially we will decide which tools we will use for development and for exporting and converting the models. Also we will choose which 3D engines and communication technologies will be used.

One of the basic necessary decisions, which have to be made is the selection of the operating system. One restriction is, that all development tools need to support it. There was a choice between *Linux* and *Windows*, since the development tools were available for both of them. Still the support of Linux was not as good it was for Windows. Also some of the tools, which were used for conversion had no support for Linux, so the decision was made, to use Windows.

For development, the use of the 4DIAC-IDE to implement the control application has already been settled by the presettings. For the simulation part, the IDE selected is Visual Studio 2005 [20]. The reason for this decision is, that Irrlicht offers best support for this version of the IDE, since the documentation and tutorials mainly focus on the 2005 version.
Since Visual Studio 2005 is already used for the simulation part, also the communication will be developed using the same IDE. The close binding of the IDE to Windows is also not a problem, since it is supported by all the used technology.

As a source for LEGO® models, several libraries exist, which provide 3D models of LEGO® bricks. For our purposes, the models will be imported from the LDraw library [15], where all of the needed LEGO® bricks can be found as models in *.ldr* format. In our experiments the models will be assembled into a virtual factory, and animated as far as the models support animation.

### 4.1.1 3D Game Engine Selection

For the simulation a state of the art, well supported open source 3D game engine will be used to display the devices and the environment. There are a lot of different 3D game engines on the market. All of the engines have some features in common, like rendering of the 3D model in real-time, means to load and display textures and sprites, display text, render special effects like fire, fog and water, and so on. Furthermore it provides means to load, display, and animate 3D models. Most of the game engines also come with support of a sound system, physic system, I/O-system, network support and data management. A list of 3D engines can be found at [1] and [8]. The features that mattered for the selection were:

- Open Source
- Good Support
- Good Documentation
- Availability
- A way to import *.ldr* models
- Being up-to-date
- Supporting collision detection

Several engines fulfilled the requirements. The two most promising of them are OGRE [23] and Irrlicht [12], which were chosen for closer evaluation.

**OGRE**

OGRE (Object-Oriented Graphics Rendering Engine) [23] is a scene-oriented, flexible 3D engine written in C++. It is designed to make it easier and more intuitive for developers to produce applications utilising hardware-accelerated 3D graphics.

The class library abstracts all the details of using the underlying system libraries like Direct3D and OpenGL and provides an interface based on world objects and other intuitive classes. OGRE provides many features, which are documented in the OGRE Wiki and the API [23]. Several tutorials are provided as well, some of which are out-dated. At the moment, when the engines were evaluated, the Wiki just underwent an update. Therefore several links just did not work. OGRE does not support direct import of *.ldr* files, containing the LEGO® models. Through a chain of conversion tools those *.ldr* files can be converted into a format, which can be loaded by OGRE. Since the conversion tools did not function properly, those models were not displayed correctly in OGRE. Trying a different format, which is supported by a plug-in, OGRE could not be compiled. Also the plug-in was just available for a certain version of Visual Studio, which is not used for this thesis. Searching for another way to import the models correctly, only more dead links and unavailable sources were found. Even trying to use the GUI plug-ins for OGRE did not work without problems.
Summarizing the evaluation of OGRE:

- Errorous tools to convert *.ldr* files into OGRE *.mesh* files
- A lot of dead links, unavailable sources, and outdated tools
- Tools only supporting a certain IDE
- Errorous GUI plugins

**Irrlicht**

The Irrlicht [12] Engine is a cross-platform high performance real-time 3D engine written in C++. It features a powerful high level API for creating complete 3D and 2D applications such as games or scientific visualizations. It comes with an excellent documentation and integrates all state-of-the-art features for visual representation such as dynamic shadows, particle systems, character animation, indoor and outdoor technology, and collision detection. All this is accessible through a well designed C++ interface, which is easy to use.

Testing the engine compared to OGRE, Irrlicht is a lot easier to handle from the beginning on. It supports direct import of several file formats, so the conversion chain only consists of one conversion (*.ldr* to *.lws*[19]). For the conversion, the conversion tool [16] can be used. The GUI system is also a part of the engine itself, so it works straight away. The documentation consists of API documentation, and several useful tutorials, which are all up-to-date. Summarizing the evaluation of Irrlicht:

- Good Documentation
- Support of several IDEs
- Direct GUI support
- Up-to-date tutorials
- No dead links found

Since the big difference in the usability of those two engines, the decision can be made, to use Irrlicht for the implementation of the framework and the experiments.

### 4.1.2 Communication Network and Synchronization

The selection of the communication technology is a bit tricky, since different communication protocols might have to be supported in the future. Especially in a communication network for a Digital Factory, different requirements might arise, because different hardware is being simulated. One of the main requirements a communication protocol for the Digital Factory has to fulfill, is that it needs to be reliable. That means that no messages may be lost, and they may not overtake each other. TCP/IP is such a reliable protocol. In offices TCP/IP is state of the art. Also in the industry TCP/IP is more and more important, and several field bus systems, which might be used in factories can be mapped to it, so TCP/IP is chosen to connect the two parts of the Digital Factory.

For the clock synchronization the same issue arises, since a lot of different technologies exist. If the Digital Factory is being executed on a single PC, then no time synchronization is needed, since there is only one clock involved. The situation, when multiple clocks need to be synchronized might arise, when parts of the system are executed on the LEGO® controller brick. Since the support of FORTE on the LEGO® controller brick is work in progress, we can only say, that an operating system will be needed in order to execute FORTE. That way probably external clock synchronization tools will be supported, depending on the results of the project.

For synchronization of the execution, we have the requirement, that the simulated system needs to act like the real one. In the real system, we have a direct cable

connection between the LEGO® Controller brick, and the sensors and actuators. This behavior also needs to be reflected in the simulation. The latency in the real system is very low, so we also need a low latency in our simulation. The use of a Socket connection on a local PC offers such a low latency. It is the lowest possible latency in our simulation framework, and therefore the approximation of the real system is "as good as it can be". Adding any middle-ware would prolong the latency. Still, if we distribute our system among several devices, which are connected by a network, then the use of a Manager to lower the latency would make sense. Such a Manager could be integrated into our framework in future work. For our purposes, the short latency provided by the execution on a single machine is sufficient.

The synchronization of the execution time is handled in an implicit way, since the coupling of the different timelines only exists, when values from the simulation part are set, or requested by the control part. We have to distinguish different scenarios:

**Set a Value** When the control part wants to set a value, only two time values have to be considered. First the transmission time, and second the set time. For our purposes, the transmission time using Sockets on a local host is a good approximation of the latency of the real system. The set time in the real system is limited by the physical attributes of the actuator. In our case, the actuator is the LEGO® servo motor, and the physical attributes are simulated. Therefore the time to set a value in our simulation is as good as it can get.

**Get a Value** When the control part requests a value, only the transmission time has to be considered. We have already discussed, that approximation of the delay using local Sockets is sufficient.

## 4.2 Implementation of the Control Part

The control part of the framework offers an interface to the control application, which is developed by students. There is one Composite Function Block for each sensor and actuator. The control part will be developed, using 4DIAC-IDE, supporting the IEC 61499 standard. In order to keep the same behavior as the real NXT Control block, the parameters of the client socket Function Block represent the interface of the NXT Control brick, and will be described now.

For the interface, the events INIT, INITO, REQ, and CNF are supported. The input variables are *QI*, which is the Event Input Qualifier, the *address* of the Server,

which will be “PORTA”, “PORTB”, “PORTC” for the actuators, and “PORT1”, “PORT2”, “PORT3” and “PORT4” for the sensors. The other parameters are sensor or actuator specific.
The individual Function Blocks of the interface will be presented in the following figures, where the input events and variables are in the top left corner, and the output variables and events at the top right corner. The center of the network, the client Function Block is situated. The interface of the Composite Function Block itself is also included into the figure at the lower right corner.

### 4.2.1 IEC 61499 Function Block Interface

The Function Block interface to the tier two simulation was designed according to the LEGO® Mindstorms NXT Controller brick, accessing its sensors and actuators. There is not only one API offering an interface to the sensors and actuators for the controller brick. For the interface in this simulation, the Lejos [22] API was chosen. Since not all of the functionality is needed, and some of the functions are just more comfortable than others, but offer the same functionality, the following interface was chosen for each brick:

**Motor:** The Function Block for the servo motor is shown in Figure 4.1. The Interface offers two event inputs *INIT* and *REQ* and two event outputs *INITO* and *CNF*. When the Function Block receives an *INIT* event, the Client Socket is initiated, and the connection to the Server Socket established. The *REQ* event triggers a send operation. Therefore only, when *REQ* is triggered, data is sent. The data inputs are *QI*, the Event Input Qualifier, the address of the servo motor, which can be the ports *A*, *B*, and *C*, and the direction of the motor, *0* meaning reversed, and *1* meaning normal direction. The *Speed* data input has the range from 0 to 100, where 0 means no rotation, and 100 means full speed. The data output of the motor Function Block is: *Status*, which displays the status of the Client Socket Function Block for debugging purposes, and *Ticks*, which are the rotations done by the motor so far. The datatype of *Ticks* is INT. The width of the *Ticks* is 16 bit, so overflow and underflow can occur.

**Lightsensor:** The Lightsensor Function Block is depicted in Figure 4.2. The Interface offers two event inputs *INIT* and *REQ*, and two event outputs *INITO* and *CNF*. When the Function Block receives an *INIT* event, the Client Socket is initiated, and the connection to the Server Socket established. The *REQ* event triggers a send operation. Therefore only, when *REQ* is triggered, data is sent. The data inputs are *QI*, the Event Input Qualifier, and the address of the Lightsensor, which can be the ports *1*, *2*, *3*, and *4*. There are no other

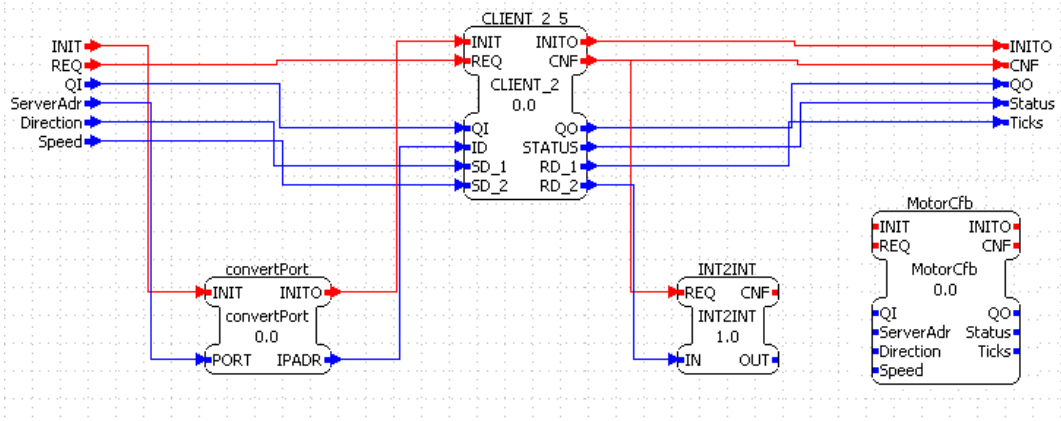


Figure 4.1: Composite Functionblock: MotorCFB

data input lines. The output *Lightval* is the value, the Lightsensor in the tier two simulation sensed. The value range is 0 to 255.

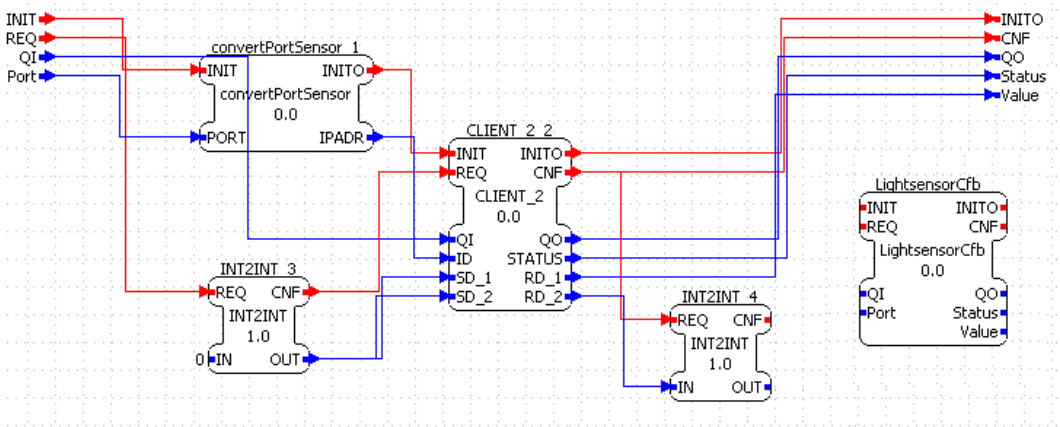


Figure 4.2: Composite Functionblock: LightsensorCFB

**Sonicsensor:** The Sonicsensor Function Block is depicted in Figure 4.3. The Interface offers two event inputs *INIT* and *REQ*, and two event outputs *INITO* and *CNF*. When the Function Block receives an *INIT* event, the Client Socket is initiated, and the connection to the Server Socket established. The *REQ* event triggers a send operation. Therefore only, when *REQ* is triggered, data is sent. The data inputs are *QI*, the Event Input Qualifier, and the address of the Sonicsensor, which can be the ports *1*, *2*, *3*, and *4*. There are not other data input lines. The output *Sonicval* is the value, the Sonicsensor in the tier two simulation sensed. The value range is 0 to 255.

**Button:** The Button Function Block is depicted in Figure 4.4. The Interface offers

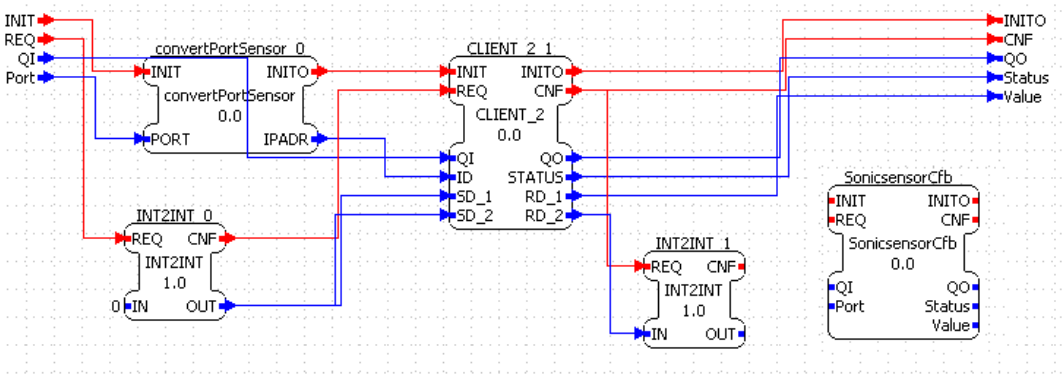


Figure 4.3: Composite Functionblock: SonicsensorCfb

two event inputs *INIT* and *REQ*, and two event outputs *INITO* and *CNF*. When the Function Block receives an *INIT* event, the Client Socket is initiated, and the connection to the Server Socket established. The *REQ* event triggers a send operation. Therefore only, when *REQ* is triggered, data is sent. The data inputs are *QI*, the Event Input Qualifier, and the address of the Button, which can be the ports *1*, *2*, *3*, and *4*. There are not other data input lines. The output *'Buttonval* is the value, the Button in the tier two simulation sensed. The value range is 0 or 1.

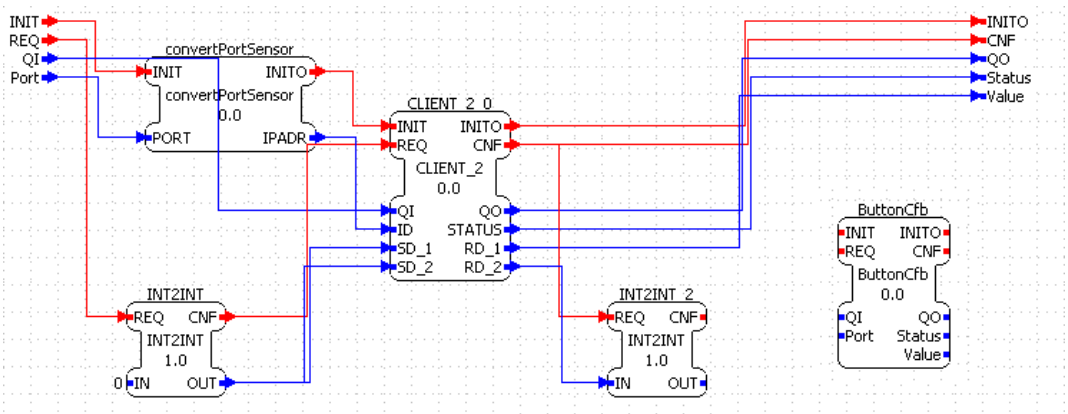


Figure 4.4: Composite Functionblock: ButtonCfb

### 4.2.2 Accessing Server Sockets

If the predefined Function Block interfaces cannot be used, then there is also the possibility to access the Server Socket of the simulation part using a *Client Function*

*Block*, as it is defined in the IEC 61499 standard (See Figure 4.5). This Client Function Block offers a TCP/IP Client Socket connection. The Interface has the event inputs *INIT* and *REQ* and the event outputs *INITO* and *CNF*. The data input is *QI*, the Input Qualifier, the *ID* of the Client, being the IP address (multicast), and the port, using the standard format: *[IP]:[PORT]*. The data output is *QO*, the output qualifier, and the *STATUS* of the Client block, giving information about the connection status. Further input and output is required, depending on the sensor or actuator, to which the Client should connect to.

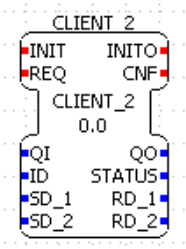


Figure 4.5: Composite Functionblock: CLIENT_2

The tier two simulation can also be used, without using Client Function Blocks at all. Using standard Sockets, compatible with Winsock 2. The ports which are used in the simulations are:

Carwash:

- Motor: 61402
- Sonicsensor: 61405
- Button: 61407

Linefollow:

- Motor1: 61402
- Motor2: 61403
- Lightsensor1: 61405
- Lightsensor2: 61406

Now that the control framework of the Digital Factory is finished, we will continue with the discussion of the implementation of the simulation framework.

## 4.3 Implementation of the Simulation Framework

In this section, the implementation of the simulation framework will be discussed. The presentation of the framework is split into three parts: *Visualization Using the 3D Game Engine*, which describes the interaction with Irrlicht, *LEGO Models in the 3D Game Engine* and the *Communication Network*, where the connection to the control part is described. Some source code will be presented in C++.

### 4.3.1 Visualization Using the 3D Game Engine

In order to use Irrlicht in a project in Visual Studio, a library needs to be added by including the *irrlicht.h* header file, and the *irrlicht.lib* library. Having done that, the application can be written, using the Irrlicht API [11]. The use of the most important classes of the API will be presented. This way the reader can gain some understanding of the functionality of the API, which is being used in the framework. In order to start using the API of the Irrlicht engine, the first thing we need to do is to create a device with the createDevice() function. The IrrlichtDevice is created by it, which is the root object for doing anything with the engine. Here also the device type is defined.

The arguments are:

- deviceType: Type of the device. This can currently be the Null-device, one of the two software renderers, D3D8, D3D9, or OpenGL.

- windowSize: Size of the Window or screen in FullScreenMode to be created. In this example we use 640x480.

- bits: Amount of color bits per pixel. This should be 16 or 32. The parameter is often ignored when running in windowed mode.

- fullscreen: Specifies if we want the device to run in fullscreen mode or not.

- stencilbuffer: Specifies if we want to use the stencil buffer (for drawing shadows).
- vsync: Specifies if we want to have vsync enabled, this is only useful in fullscreen mode.
- eventReceiver: An object to receive events.

In the framework the function was called with the parameters:

```
IrrlichtDevice *device = createDevice(EDT_DIRECT3D9, core::
    dimension2d<u32>(1024, 768));
```

Having the IrrlichtDevice, one gains access to the VideoDriver, SceneManager and the GUIEnvironment. The VideoDriver is is one of the most important interfaces of the Irrlicht Engine. All rendering and texture manipulation is done using its interface. The irr::scene::ISceneManager interface provides a lot of powerful classes and methods to support easy integration.

The SceneManager handles the scene nodes. A scene node is a node in the hierachical scene graph. All scene nodes can be created only here. There is a continuously growing list of scene nodes for lots of purposes: Indoor rendering scene nodes like the Octree *addOctreeSceneNode()* or the terrain renderer *addTerrainSceneNode()*, different Camera scene nodes *addCameraSceneNode(), addCameraSceneNodeMaya()*, scene nodes for Light *addLightSceneNode()*, Billboards *addBillboardSceneNode()* and so on. Every scene node may have children, which are other scene nodes. Children move relative to their parents position. If the parent of a node is not visible, its children won't be visible, too. This way, for example, it is easily possible to attach a light to a moving car or to place a walking character on a moving platform on a moving ship [11]. The SceneManager is also able to load 3D mesh files of different formats.

The GUI Environment is used as a factory and manager of all other GUI elements. The pointer to those interfaces is retrieved as listed here:

```
IVideoDriver* driver = device->getVideoDriver();
ISceneManager* smgr = device->getSceneManager();
IGUIEnvironment* guienv = device->getGUIEnvironment();
```

The next thing needed is a Camera Scene Node. The whole scene will be rendered from the cameras point of view. Because the ICameraSceneNode is a SceneNode,

it can be attached to any other scene node, and will follow its parents movement, rotation and so on. In our framework, two CameraSceneNodes will be used. One is a common Camera Scene Node, the other is a FPS Camera Scene Node. The difference is, that the FPS Camera Scene can be controlled, like it is common in First Person Shooters. That means, that the camera can be moved, using the buttons "W", "A", "S", "D", and the mouse to look around freely. The common Camera Scene Node has no "built-in" control functions, and the position and rotation need to be set by the application. Now still the question arises, why two cameras are needed. Since the application should support free user-controlled camera movement, and GUI Interaction, two cameras become necessary. In the FPS mode, no cursor is visible, and the mouse-movement is used to rotate the camera. This way the GUI elements could never be selected, and adjusted. For that reason, the FPS Camera is only active, when a certain button is being pressed. If the button is not pressed, the normal camera is being used, and the mouse can be used to control the GUI elements. The two cameras are initialzied in the following way:

```
scene::ICameraSceneNode* cameraFPS=smgr->
    addCameraSceneNodeFPS();
scene::ICameraSceneNode* camera=smgr->addCameraSceneNode();
smgr->setActiveCamera(camera);
```

With the setActiveCamera function, the camera, which should be used, is being set. The "receiver" is a MyEventReceiver object. Also the cursor needs to be set to visible and invisible, depending which camera is being used. In the line follower application the FPS camera can be controlled freely, and the other camera always follows the car. This way, the user can switch between two viewpoints, and set the FPS camera to a point, where the simulation is of particualar interest, and watch the simulation freely. Else the camera just follows the car all the time and the user gains a good view of the functionality.

```
if(receiver.IsKeyDown(irr::KEY_KEY_W)){
  smgr->setActiveCamera(cameraFPS);
  device->getCursorControl()->setVisible(false);
}
else{
  smgr->setActiveCamera(camera);
  device->getCursorControl()->setVisible(true);
}
```

The next thing needed is some light for the scene, so the objects can be seen. One light scene node is being added, next to the ambient light, so the objects can cast shadows as well. The LightSceneNode is added with:

```
ILightSceneNode* light1 = smgr->addLightSceneNode( 0, core::
    vector3df(0,4000,0), video::SColorf(0.3f,0.3f,0.3f)
    ,50000);
```

setting it to the coordinates (0,4000,0) with the radius 5000 and the RGB colorvalues (0.3,0.3,0.3).

Collisions in general are controlled by the SceneCollisionManager. The Scene Collision Manager provides methods for performing collision tests and picking on scene nodes. It is used in the following way:

```
scene::ISceneCollisionManager* collMan = smgr->
    getSceneCollisionManager();
```

Some of the sensors need the function to select another SceneNode. For example the light sensor must know, what it is "looking" at. To support that function, a TriangleSelector is needed. Using a ray, which is totally user defined, an intersection with the target can be produced, and that way, a SceneNode can be selected. The selection is made by analyzing the data of the collision with the ray. It contains a reference to the SceneNode, to which the triangle belongs to. The following call is all that is needed to perform ray/triangle collision on every scene node that has a triangle selector. It finds the nearest collision point/triangle, and returns the scene node containing that point. Irrlicht provides other types of selection, including ray/triangle selector, ray/box and ellipse/triangle selector, plus associated helpers. The collRay is the line, where the collision is detected.

```
scene::ISceneNode * selectedSceneNode = collMan->
    getSceneNodeAndCollisionPointFromRay(
collRay,
intersection, //This will be the position of the collision
hitTriangle, //This will be the triangle hit in the
    collision
IDFlag_IsPickable,//This ensures that only nodes that we
    have set up to be pickable are considered
0); //Check the entire scene
```

### 4.3.2 LEGO® Models in the 3D Game Engine

Describing LEGO® bricks and models in a virtual environment is a challenging prospect. In 1995 James Jessiman developed just such a system with his LDraw program and file format. Since then the LDraw file format has grown to be the standard by which most create LEGO® models on the computer. For the 3D models, which will be used in the 3D Game Engine, the LDraw [15] library is used as a source. It supplies many LEGO® bricks as digital 3D models. With the tool LDView, the models can be viewed. The tool is shown in Figure 4.6.

The LDraw file format cannot be used in Irrlicht directly. Therefore it has to be converted. The tool "ldraw2lws" [16] offers a conversion of LDraw files to Lightwave Objects (.lwo) and Lightwave Scenes (.lws). Those files can be used in Irrlicht, and loaded into SceneNodes. How those SceneNodes are created, is shown with the example of the motor model, where first the .lwo object is loaded into a Mesh object, and then the SceneNode is created out of the mesh.

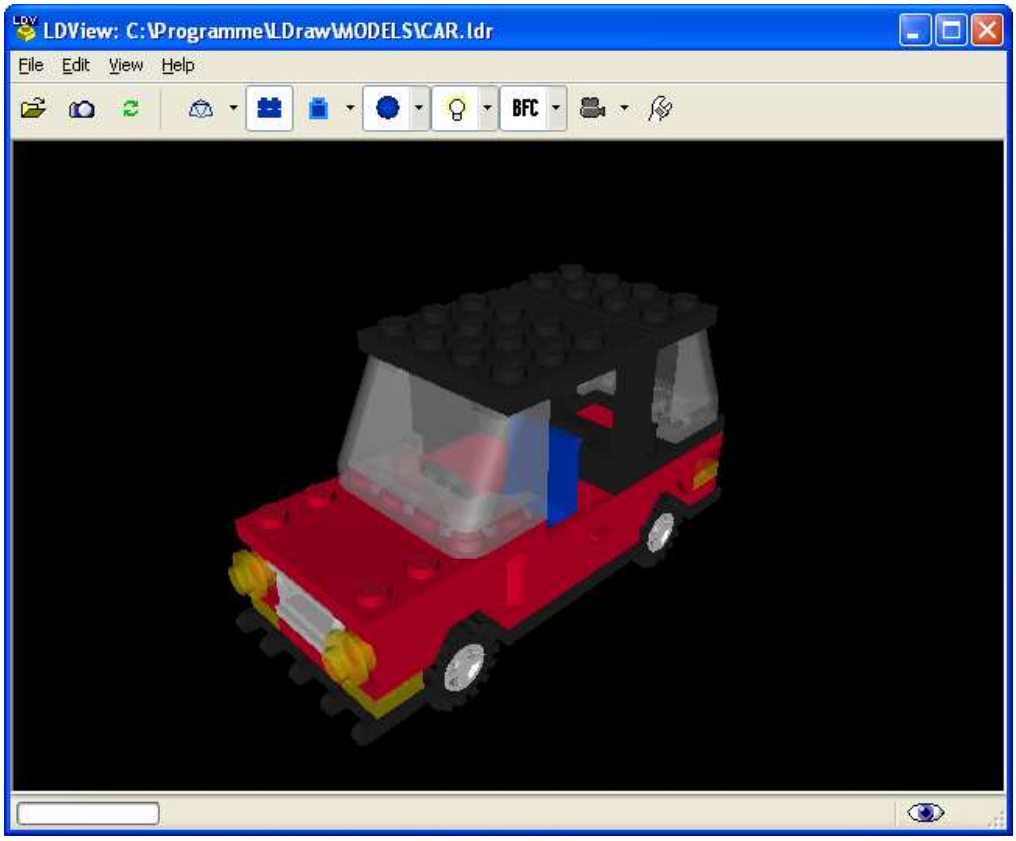


Figure 4.6: LEGO® model displayed in the LDView tool

```
IAnimatedMesh* motorMesh = smgr->getMesh("../../media/
    motorcomplete.lwo");
  if(!motorMesh){
    device->drop();
    return 1;
  }
  IAnimatedMeshSceneNode* NXTNode = smgr->
      addAnimatedMeshSceneNode(NXTMesh);
  if(NXTNode)
  {
    NXTNode->setScale(vector3df(100,100,100));
    NXTNode->setPosition(vector3df(300,150,220));
    NXTNode->setRotation(vector3df(270,0,0));
  }
}
```

The simulation part of the Digital Factory requires accurate representation of the LEGO® bricks. Therefore the attributes were investigated, and mapped as accurate as possible. One important example of the physical attributes is the rotation speed of the servo motor. It was programmed according to the following representation. The rotation speed of the motor under load in respect of the input speed from zero to full speed according to [18] is shown in Figure 4.7.

The colors represent following settings:

- Dark blue: not loaded, 9V NXT power
- Green: not loaded, 7.2V batteries
- Pink: 11.5 Ncm load, no Power Control, 9V NXT power.
- Yellow: Loaded with a 11.5 Ncm, 9V NXT power. This curve shows the efficiency of Power Control: up to 70% the speed is the same as an unloaded motor.
- Blue: Loaded with a 11.5 Ncm, 7.2V NXT power. Up to 50% the speed is the same as a no-load, 9V powered motor (and actually faster than a 7.2V powered motor without Power Control!).

The following code was used to retrieve the rotations out of the value received by

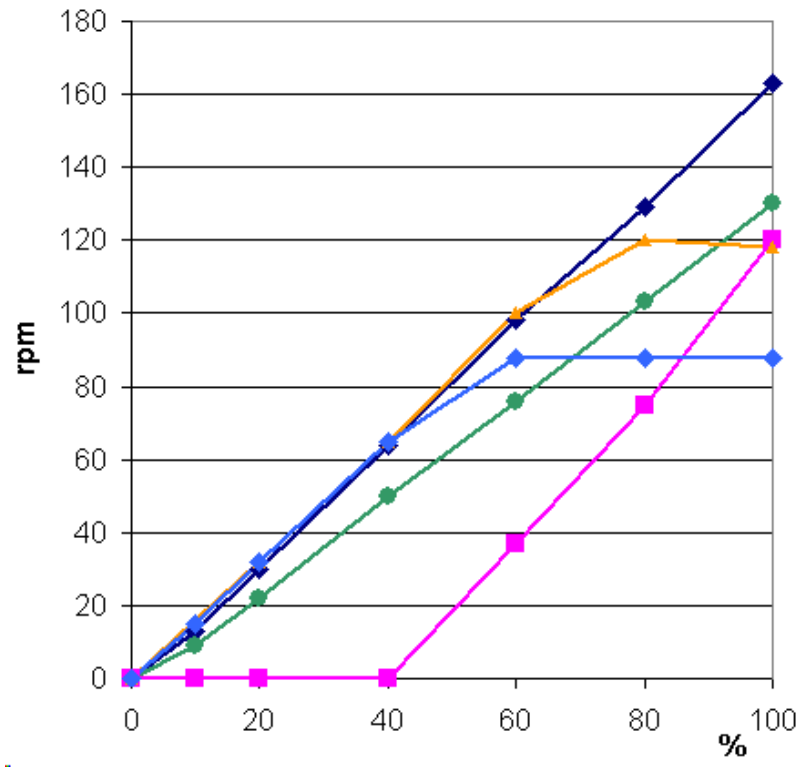


Figure 4.7: LEGO® Mindstorms NXT Motor Speed [18]

the control application. The parameter motorSpeed can have values between 0 and 100.

```
float calcMotorRPS(float motorSpeed){
  if (motorSpeed < 60)
    return (motorSpeed*100)/60;
  else if (motorSpeed < 80)
    return motorSpeed+40;
  else
    return 120;
}
```

### 4.3.3 Communication Network

The communication was realized using server client, TCP/IP socket connections. Each sensor and actuator has its own connection, playing the server part. In the

control application, the TCP/IP clients can access any of the sensors and actuators. For the server socket, Windows Sockets 2 [30] was used. For each server a new thread is created, so the communication can be handled without depending on the execution speed of the simulation application. The server threads are created in the following way: The pDataArray is a void pointer, which can contain parameters for the thread. The dwThreadIdArray will contain the ID of the thread, which is assigned by the CreateThread function. The returnvalue of the CreateThread function, is the handle to the thread.

```
1  void restartServerThreads (PMYDATA pDataArray[],
2    DWORD dwThreadIdArray[],HANDLE hThreadArray[]){
3    // Create MAX_THREADS worker threads.
4    int i=0;
5    for(i=0; i<MAX_THREADS; i++ )
6    {
7      hThreadArray[i] = CreateThread(
8        NULL,
9        0,
10       serverThread,
11       pDataArray[i],
12       0,
13       &dwThreadIdArray[i]);
14     if (hThreadArray[i] == NULL)
15       ExitProcess(i);
16   }
17 }
```

For creating the socket connection, the following steps were taken:[1]

1. Winsock has to be initialized.

2. The server address and port are resolved.

3. Create a socket, so that a connection can be established.

4. A listening socket is created, with the option to reuse the address, and the “do not linger” option turned on, to enable fast restarting of the threads.

5. Accept a client connection.

[1] The source code was skipped here, the reference can be found at [30]

Then, with the listen socket set up, the connection is established, for the sixth step, the data exchange. The servomotors are assigned to PORTA, PORTB and PORTC, and the sensors from PORT1, to PORT4. In order to see, if data is exchanged, a debugmessage is created, displaying the state of the transmissions:

```
// Receive until the peer shuts down the connection
do {
  iResult = recv(ClientSocket, recvbuf, recvbuflen, 0);
  if (iResult > 0) {
  EnterCriticalSection(&CriticalSection);
  debugMsg = "";
  debugMsg += L" port:";
  debugMsg += port;
  LeaveCriticalSection(&CriticalSection);

  for(int i = 0;i < recvbuflen; i++){
    sendbuf[i]=recvbuf[i];
  }

  if(strcmp(PORTA,port)==0){
    sendbuf[0]=67&(0xFF<<0);
    EnterCriticalSection(&CriticalSection);
    motorDir = recvbuf[2]&(0x01);
    motorSpeed = recvbuf[5] &(0xFF);
    if(motorSpeed >100){
      motorSpeed = 100;
    }
    sendbuf[1]=0&(0xFF<<8);
    sendbuf[2]=((int)motorTicksA)&(0xFF<<0);
    motor1sending = true;
    LeaveCriticalSection(&CriticalSection);
  }
  //...


  iSendResult = send( ClientSocket, sendbuf, iResult, 0 );
  if (iSendResult == SOCKET_ERROR) {
      printf("send failed: %d\n", WSAGetLastError());
      closesocket(ClientSocket);
      WSACleanup();
      return 1;
  }
```

```
38    else if (iResult == 0)
39        printf("Connection closing...\n");
40    else  {
41        printf("recv failed: %d\n", WSAGetLastError());
42        closesocket(ClientSocket);
43        WSACleanup();
44        return 1;
45    }
46
47  } while (iResult > 0);
```

In the end, connection is closed, and all remaining references are cleaned up.

## 4.4 Summary

In this chapter we have seen how the framework of the Digital Factory was implemented. We started with the selection of the development tools, and made design decisions, which 3D engine to use, which source to take for the 3D models of the LEGO® parts, and how the communication should work.

Then the implementation of the control part of the Digital Factory was described, giving detailed information about the interface, which can be used by the students in their own applications. The individual Function Blocks of the interface were listed, which represent the LEGO® sensors and actuators. Also the communication mechanism was described, so that the simulation part can be used without the offered interface.

The implementation of the simulation framework was presented, providing some source-code examples of the interaction with the 3D game engine. The loading and animation of the 3D models was described. Also the communication system, using a threaded TCP/IP Socket connection for each simulated sensor and actuator was discussed in detail.

We can conclude, that all parts of the Digital Factory could be implemented, and the framework realized. The use of the interface in the experiments, and the results will be discussed in the next chapter.

# 5 Discussion of Results

In this section, the findings of the project will be presented. Previously we created a framework for a Digital Factory. In order to test this framework, two applications are developed, coming from different fields of the automation sector. Those applications are designed according to the two tier model of the Digital Factory. The results will tell, how well a 3D game engine and a IEC 61499 based controller can be used to develop useful simulations of a factory.

The first experiment is a car wash station, and the second experiment is a robot, which follows a line. Both applications use the framework of the Digital Factory. Now we will start with the discussion of the framework, which has already been implemented, present the experiments, and discuss the results.

## 5.1 Experiments

In this section we will deal with the two experiments. The section will be split into two parts, representing one experiment each. The experiments are a simulation of a car wash station, and a robot which follows a line. But first we will have a look at what we can use for the implementation of the experiments.

The framework that we created offers an interface to create a Digital Factory, which consists of the control, and the simulation part. For the control part the interface is easy to use. The Function Blocks for the sensors or actuators need to be included into the application, and by connecting them with the own application, the interaction with the simulation part works. We can use the provided Function Blocks for the servo motor, the light sensor, the ultrasonic sensor, and the button, and use them to build an arbitrary application.

The use of the simulation part of the factory is not that easy. Since we lack an editor, we need to create the simulation application in the IDE. The integration of an editor would be a topic for future work. But also with the IDE the implementation was

quite modular, since the physical attributes of the sensors and actuators do not change from experiment to experiment. Only the arrangement and the interaction of the sensors and actuators changed, and need to be connected.

### 5.1.1 Car Wash Station

The car wash station consists of three LEGO® Mindstorms NXT parts: Servo motor, light sensor, and a button. The car is made out of boxes, which can be configured by the user to form the shape of a car. This "car" is being moved through the car wash station, which consists of a brush represented by a LEGO® brick. The brush is being moved by the servo motor. The distance between the brush and the car can be measured by the ultrasonic sensor, which is used for that purpose. The value of the ultrasonic sensor can be used by the control application to set the value of the motor. An overview of the setup can be seen in Figure 5.1.

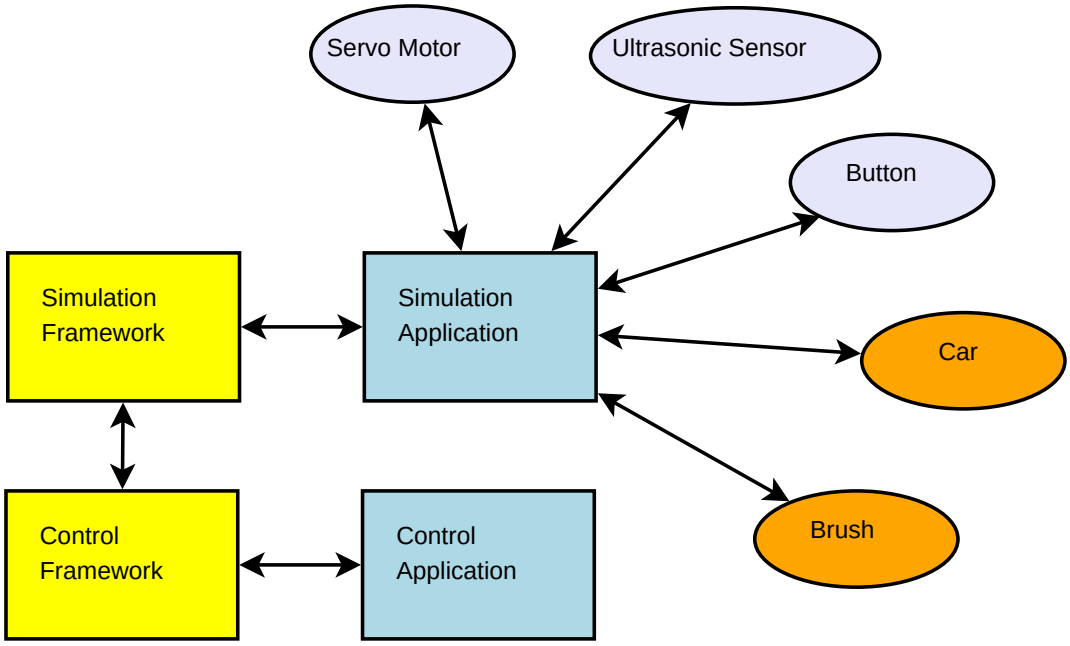


Figure 5.1: An overview of the car wash application

**Simulation Part** The use of the sensors and actuators in the simulation part of the car wash station was quite easy, since the physical attributes were already settled. The interaction between the parts was more challenging. For example the gear transmission between the servo motor, and the movement of the brush

had to be settled. The speed was measured in units per second, where one unit is one pixel in the 3D engine.

The user can define the shape of the car, which is being displayed by several boxes in a row, which are connected. The user can adjust the height of the connection point of the boxes, so the shape of a car can be created. Also the speed of the car can be adjusted.

The brush in the car wash station is represented by a LEGO® bar. The bar is being moved according to the speed of the motor, and the adjusted transmission. When the bar is continuously moved up, at one point it reaches the button. Then the position of the brush is known. This enables the control part to execute a "homing run". The ultrasonic sensor measures the distance from the ground. If the car is between the sensor, and the ground, the distance is shortened. A screenshot is shown in Figure 5.2


Figure 5.2: A screenshot of the car wash application

In the screenshot, the elements of the simulation can be seen. The sensors and actuators are a servomotor, the sonic sensor, and the button. Also the

NXT controller can be seen in the background. The car, which is created out of five boxes is shown in the initial position. Also the motor is in the initial position, resulting in the brush (the brick with the red stripes) being in the lowest position. With the slide in the top left corner the user can define the speed of the car. In the top right corner, a status box informs about the status of the servers, and the connection. In the Logging ListBox, collisions are displayed, and the distance of the object to the sonic sensor, if an object is detected by it. In the top middle of the screen we can see a bar with "Brush:" and "Ticks:", which displays information about the height of the Brush, and how many ticks the servomotor has done. If the car hits the brush, a collision is detected, and a fire animation activated, which is displayed in Figure 5.3

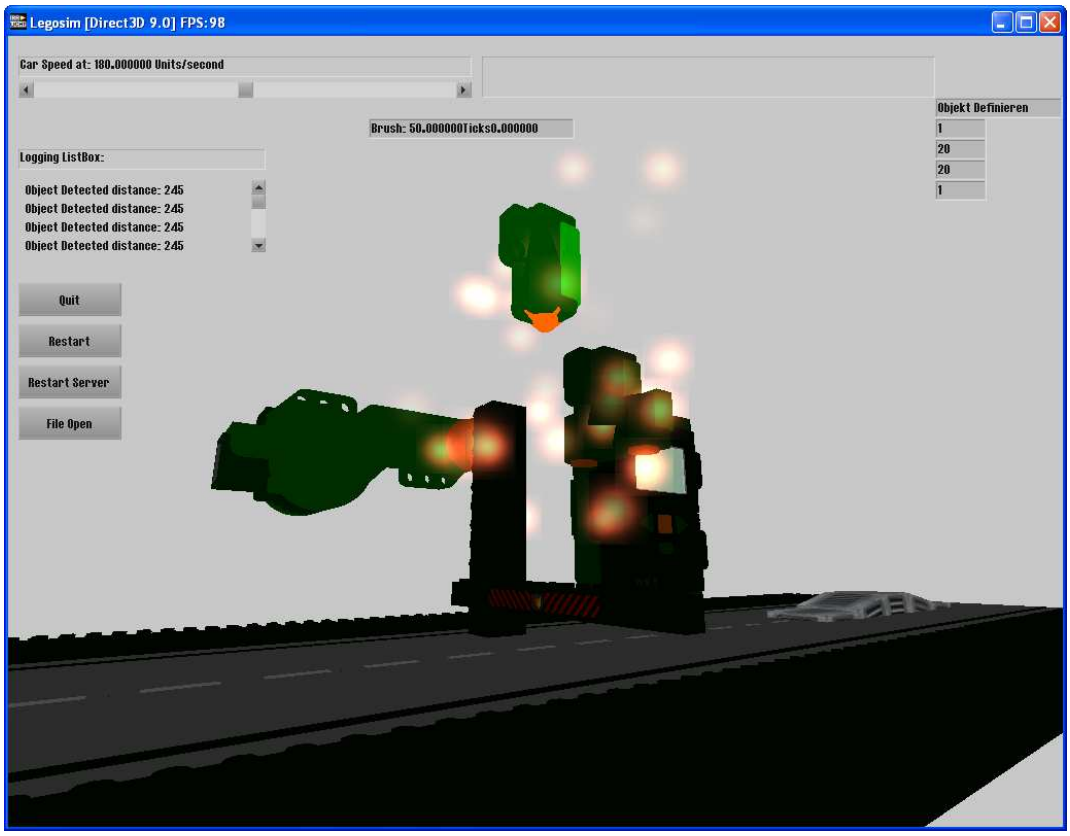


Figure 5.3: The car crashed into the brush

The Restart button resets the simulation, putting the car, the brush, and the servomotor into the initial position. The resetting of the car works the following way. First the animators of the individual boxes are removed, and the fire animation stopped. Then a new animator is created for the boxes,

which sets them to their original position. In the last step, the flags, that indicate a crash and the restart of the simulation are reset.

```
if(restartBtn == true){
    for(int i =0; i< (VERTICES_NR-1); i++){
      sNodes[i]->removeAnimators();
    }
    ps->setParticleSize(core::dimension2d<f32>(0.0,0.0))
        ;
    listbox->clear();
    for(int i =0; i< (VERTICES_NR-1); i++){
      scene::ISceneNodeAnimator* anim = smgr->
          createFlyStraightAnimator(Middles[i],Middles[i
          ]+core::vector3df(-1000,0,0), s32(setAnimSpeed)
          , false);
      if (anim)
      {
        sNodes[i]->addAnimator(anim);
        anim->drop();
      }
    }
  }
  crashed = false;
  restartBtn = false;
}
```

The gear transmission between the motor and the brush was set to a ratio of 2/3 of the motor speed. Since the motor can operate at 120 rotations per second at full speed, the brush is being moved by 80 units (0.8 units per rotation). The distance between the lowest and the highest position of the brush is 250 units. Therefore the brush can be moved from the bottom to the top within 3.125 seconds, if the motor spins at full speed. For the communication connection, for each sensor and actuator a thread with a socket server is started. Those connections can be used by the control application to set and get values from the sensors and actuators.

**Control Part** Creating the control application we could use the interface offered by the control part of the Digital Factory framework. That way a basic control application was implemented. In the car wash application the value of the ultrasonic sensor is polled activated every 3ms. The REQ event is generated by the E_CYCLE Function Block. The values returned by the sonic sensor is fed into a controller block, which compares the current value with the desired

value, and forwards the result to the motor Function Block. The controller block is a basic Function Block, with an interface depicted in Figure 5.4, and the following algorithm, for mapping the input to the output:

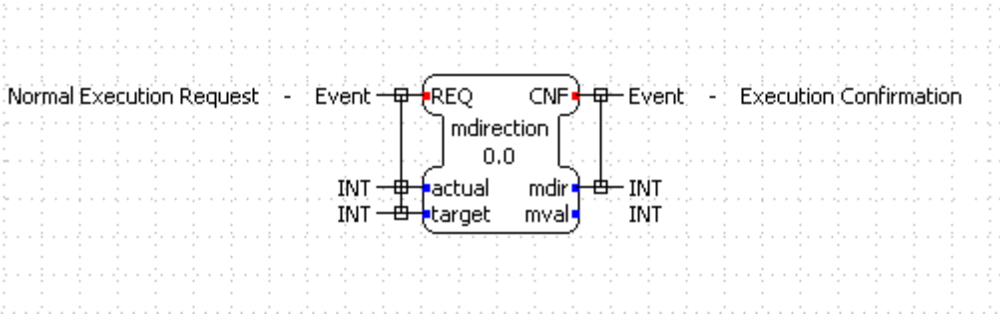


Figure 5.4: The Mdirection Basic Function Block

```
error:=actual-target;
IF error >= 0 THEN
   mval:=error;
   mdir:=1;
END_IF;
IF error < 0 THEN
   mval:=-error;
   mdir:=0;
END_IF;
```

The motor CFB gets the results (motorspeed and -direction) of the controller block, and returns the Status of the Client Function Block, as well as the ticks, the servomotor has already take. This value is transformed into a height equivalent value, which can be compared to the sonic value in the controller block again, forming a control loop. The Function Block network is depicted in Figure 5.5.

### 5.1.2 Line Follow Robot

The line follow robot application consists of two servo motors and two light sensors, which together form the robot. The light sensors can distinguish bright and dark sections on a map, which is provided by the user. This value can be sent to the control application, which processes the light value, and produces the target motor speed according to the value. An overview of the setup is given in Figure 5.6.

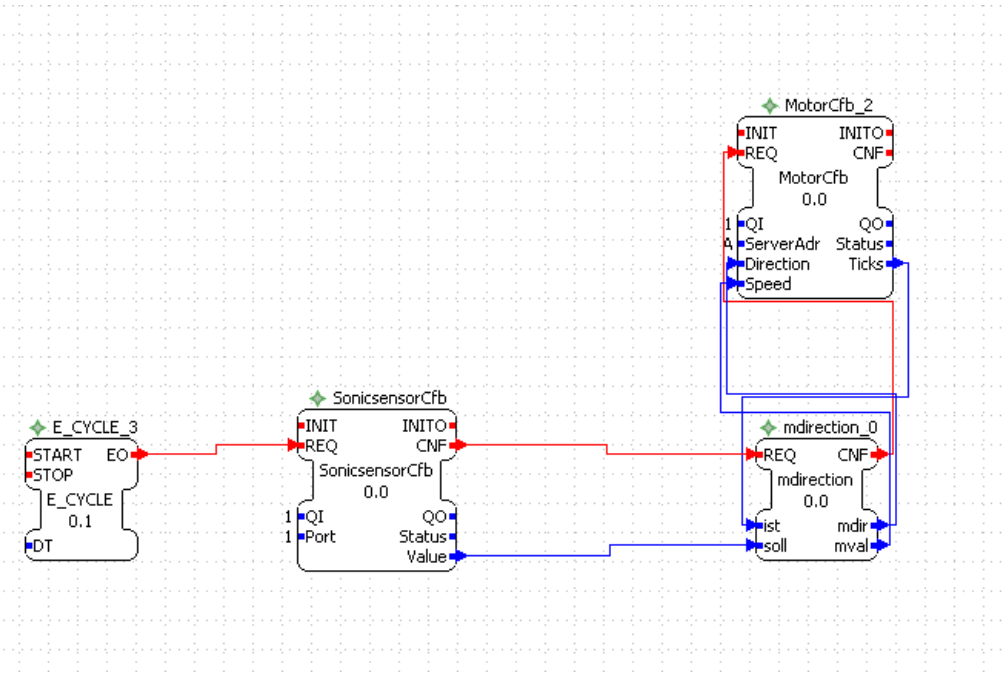


Figure 5.5: The network of the car wash application

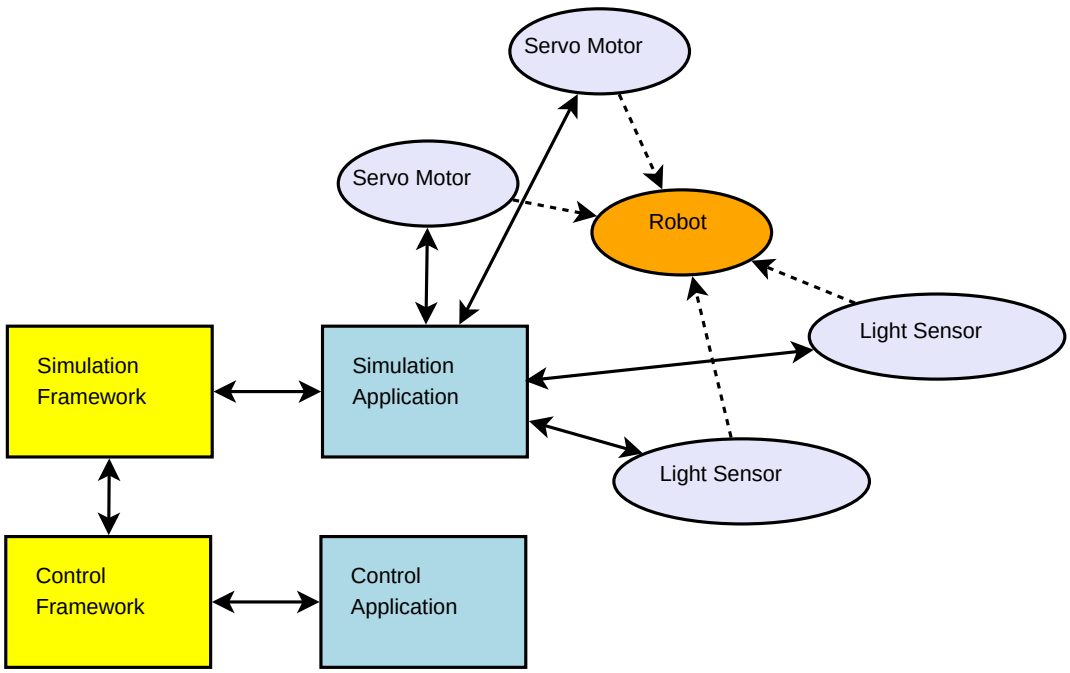


Figure 5.6: An overview of the linefollow application

**Simulation Part** The line follow simulation enables the user, to design a control program, which enables a "car" robot to follow a line. The car consists of two light sensors and two servomotors. A screenshot of the simulation can be found in Figure 5.7. The simulation is well suited for robotic simulations. The motor speed of each wheel can be controlled separately, by the tier two simulation. The values of the two light sensors are sent to the control application. The user has a number of values that can be adjusted in the GUI.

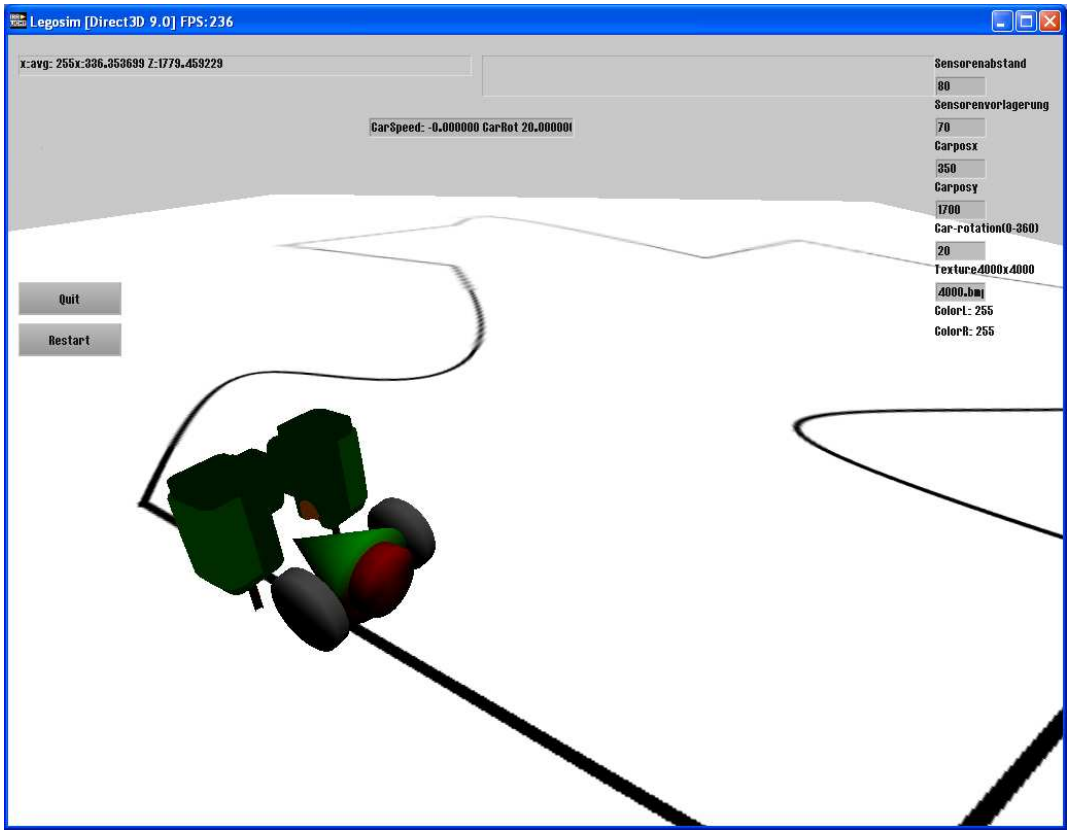


Figure 5.7: A screenshot of the linefollow application

- The distance between the lightsensors.
- The distance between the axis of the wheels and the axis of the lightsensors.
- The initial position of the car.
- The initial orientation of the car.

- The texture of the line that the car will follow.

The texture of the line is an image, drawn by the user. This image is mapped onto a box. The values of the light sensors are displayed in the GUI as well, named ColorL and ColorR. The values are an average of the RGB values in the image. The simulation works best, when the light sensor is close to the surface. For greater distances, the area the light sensor "sees" becomes too big, and the line cannot be sensed well. In the simulation, the servomotors are not shown. The physical attributes are calculated manually, and are not retrieved with a physics simulation, because the overhead would be too big. The values displayed in the GUI are the speed of the car, the rotation of the car, a status display of the servers and the connection, and the quit button. The following source code is used to calculate the value of the light sensor in the simulation part. First the position of the sensor is calculated, then the pixel selected, and finally the color value is assigned to the variable:

```
1 vector3df positionL=tmpCylNode->getAbsolutePosition();
2 vector3df positionL1=tmpCylNode1->getAbsolutePosition();
3
4 SelectedPix = imgTexture->getPixel(positionL.Z,positionL
    .X);
5 lightsensor = (SelectedPix.getBlue()+SelectedPix.getRed
    ()+SelectedPix.getGreen())/3;
6 SelectedPix = imgTexture->getPixel(positionL1.Z,
    positionL1.X);
7 lightsensor1 = (SelectedPix.getBlue()+SelectedPix.getRed
    ()+SelectedPix.getGreen())/3;
```

**Control Part** The control part of the line follower application is shown in Figure 5.8. The network consists of a E_CYCLE, defining how often the tier one simulation should be polled, two motor CFB, and two light sensor CFB. The light sensor's value is zero, if it senses a black line, and 255, if it senses white. Its value is being transformed into the same value region of the motor CFB, which is between 0 and 100. Then the transformed output can be fed into the motor block directly. That way, the wheel, spins at full speed if the sensor on the same side senses white. If the sensor senses black, the value fed into the motor CFB is zero, which results in stopping the motor.

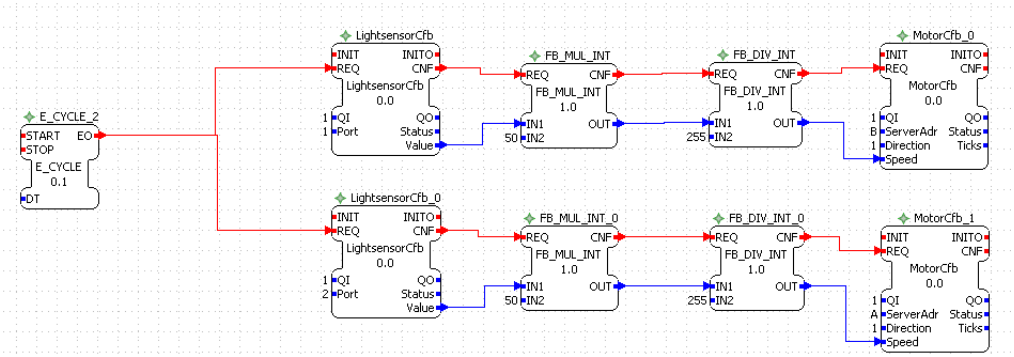


Figure 5.8: The Function Block network of the line follow application

## 5.2 Discussion of the Framework

The framework has already been implemented in the last chapter. In this section we will discuss, whether all desired requirements could be reached, and which are not included in the framework. Let us start with the features, which were required, and successfully implemented:

- The two tier concept of the Digital Factory is supported by the framework (control and simulation part).
- The control part consists of an IEC 61499 framework, which is executable in FORTE.
- An interface is provided, so that IEC 61499 control applications can interact with the control framework.
- The simulation part consists of a simulation framework, which simulates the LEGO® hardware (servo motor, light sensor, ultrasonic sensor and the button).
- This simulated hardware can be controlled by the control application from the control part.
- The simulation is a serious game, using a 3D game engine, where the LEGO® hardware 3D models are displayed, animated, and can interact.
- Visual feedback is given, and a user interface provided, so that parameters of the simulation can be set.

- Within that framework experimental setups are possible, where facilities can be simulated.
- The LEGO® hardware is simulated in a physically correct way.
- The behavior of the simulated models needs to be the same as of the real hardware.
- Students are able to create a control application, which can access the simulation part, and control the simulated devices.
- The simulation and control applications run synchronized, and in real-time.
- The principle of Soft Commissioning® is supported.

We can see, that all the requirements out of the *Concept* section have been reached. But also some parts, which could have been implemented, are not contained in the framework:

- There is no editor to create a simulation environment.
- The communication network supports one technology, and no field bus communication.
- A middle-ware could have implemented in the distributed simulation, so that the latency would be lower for certain application areas, but was not.

Those parts would enhance the functionality of our Digital Factory framework, but were not needed for our purposes. Still for future use, they might be useful. All the features which were not implemented are out of the "nice to have" section, so no essential part is missing in our framework. Now we will come to the experiments part, where the framework is used to create two experiments.

## 5.3 Discussion of the Experiments

With the two simulations, the user can gain first contact with the concept of the Digital Factory. The big benefit is, that the control application can be developed and tested, before the car wash exists in reality. Therefore the whole process is sped up, since a tested control application is already available, when the whole system goes

live, which is the concept of Soft-Commissioning and Virtual Start-up. Still not all the possible situations can be simulated, since the human factor cannot be modeled easily. For example, in the car wash station, the person, sitting in the car, could start the engine, and drive forward, which was not considered in the simulation.

Still we can provide two working experiments, which can also be used for teaching purposes. Even if there is no editor to create new facility setups, the two existing simulation applications can already be used, to create multiple control applications, which can be tested on an ordinary PC. The effort to create a new facility depends on the experience of the user. The necessary steps are, to create a Node and set the position and orientation for each Model, which will be used. Also the interaction between the models need to be defined. For example, the rotations per second of the motor can be accessed, but the influence on the model needs to be set manually. For the sensors, the environment, which should be sensed needs to be created. All the communication does not depend on the setup of the facility, just the port needs to be set, at which the sensor or actuator is connected to the control application.

## 5.4 Summary

In this chapter the results of the framework for the Digital Factory, and the two experiments were discussed. The question was answered, how well the framework was suited to meet the requirements of the Digital Factory, and if the experiments were any useful. We have seen, that an easy to use interface to the framework could be provided at the control part. This interface could easily be used in the experiments as well. At the simulation part, an editor to create the virtual facility was not included in the framework, since it would have exceeded the bounds of this thesis. Still the requirements, which were defined in the concept were met, but several more functions could be added to the framework and the experiments. This will be the topic of the next chapter, the Outlook.

# 6 Outlook

We have seen, that 3D game engines are well suited for simulation of factories. In order to realize simulations of more complex plants, physic simulation engines would be handy, since a lot of physical attributes, degrees of freedom, and interaction between the devices have to be computed, and tracked. In a bigger setup, manual simulation of the attributes would result a lot of overhead, and is also prone to errors. Irrlicht supports plug-ins for physic simulation. The models used in the engine would have to be composed of several parts, which support animations, so that each movable part can be animated according to their physical interaction.

For the simulation part of the Digital Factory, an editor would be useful, in order to enable the students to generate own facility setups. That way, the students could define own plants in the editor, and test them with the control application, which have been created by them as well. The editor would need to have a graphical interface, in which the user can select components, place them in the virtual space, and connect them. The challenge would be to design the tool easy enough to use, so that the systems can be created in an easy way, without having detailed plans of the plant. For example cogs would have to be "magnetic", so that they automatically connect in the right way, once they are brought together close enough. In order to create complex plants, like assembly lines, also objects have to be designable. Even more, the parts, of which the object consists of, have to be designable, and combined to the whole product subsequently.

Furthermore, next to the real-time execution, faster and slower execution could be added, so the user can gain more feedback, and survey the simulation closely, or just "fast forward" to the end of the production. That would also add requirements to the communication network, since the data transmission has to be faster as well, if the simulation is run faster.

In the communication layer, it would be handy to be able to switch between distributed communication, and local communication using shared memory. Depending on the application, and the system which is being simulated, this would result in possibly having more realistic temporal behavior. Also different communication protocols could be supported, in order to cover more simulation setups. According to

the different requirements of the different simulations also certain synchronization mechanisms would have to be supported.

# 7 Conclusion

Within this thesis the question was discussed, whether a 3D game engine can be used, to develop a distributed simulation framework of LEGO® Mindstorms NXT based plants. The distributed simulation was designed according to the concept of the Digital Factory, which is split into a simulation part, and a control part. By a first approach the requirement was identified, that 4DIAC-IDE will be used for the development of the IEC 61499 control part, and FORTE as the runtime environment. The 3D models of the LEGO® Mindstorms NXT sensors and actuators are provided by the Ldraw library. Also we identified the need of a communication network, which connects the control, and the simulation part of the Digital Factory.

From the research of the state of the art, we learned, that Soft Commissioning® and Virtual Start-Up can be a part of the Digital Factory. Both describe the transition from the virtual to the real factory, where the the simulation part is controlled by a Soft-PLC on the one hand, and by the real PLC on the other hand. For the simulation part, the key features of a 3D game engine were introduced, and our project identified to belong into the category of serious games. The basis of the Digital Factory is a distributed simulation framework, where several problems, especially with correct timing and clock synchronization need to be solved. For the communication network several technologies exist, which can be mapped to TCP/IP.

Based on the state of the art, a concept could be developed. First, the current state of the environment, without our project, was compared to the desired state, where the project was included. That way some requirements were found, which were extended during the development of a Use-Case Diagram. The complete list of requirements was categorized into need to have, nice to have, and not to have. A Class Diagram supported the system design, where the interaction of the classes was defined.

Using the requirements and the system design, the framework of the Digital Factory could be implemented. For the implementation, the development environment had to be chosen, which included the selection of the IDE, which 3D game engine should be used, and the selection of the network technologies. With the development en-

vironment set up, the control part and the simulation part of the framework of the Digital Factory could be implemented.

Based on the requirements, which were defined in the concept section, the implemented framework was inspected. That included a comparison between the defined, and the implemented requirements, as well as a discussion of the features, which could have been included, but were not. In order to test the framework, two experiments were developed. The first was a simulation of a car wash station, and the second was the simulation of a robot, which follows a user defined line. The results of the experiments were discussed as well.

Finally we had a closer look at the features, which might be subject to future development. The integration of a physic simulation, and an editor to create facility setups, would be possible, as well as the integration of further communication technologies into the communication network.

On the whole, the answer to the question, whether a 3D game engine can be used, to develop a distributed simulation framework of LEGO® Mindstorms NXT based plants is positive, which can be confirmed by the results of this thesis.

# Bibliography


[1] 3d Engines Overview, Online Available: `http://www.devmaster.net/engines/`, Nov. 2010.

[2] 4DIAC, Online Available: `http://www.fordiac.org/`, Oct. 2010.

[3] A. Hu and S. D. Servetto, "Algorithmic aspects of the time synchronization problem in large-scale sensor networks," Kluwer Academic Publishers, pp. 491–503, 2005.

[4] M. Adams, *SPS/IPC/DRIVES 2004*. Berlin: FRANZIS, 2004, vol. Elektrische Automatisierung, Tagungsband, ch. "Start of Production" = Termination Point of the Digital Factory?, pp. 245–253.

[5] Alois Zoitl, *Real-Time Execution for IEC 61499*. OOONeida, 2009.

[6] C. Constantinescu, V. Hummel, and E. Westkämper, "Fabrik life cycle management," Werkstattstechnik online 96 (2006), pp. 178–182, 2006.

[7] Fuber, Online Available: `http://sourceforge.net/projects/fuber/`, Nov. 2010.

[8] Game Middleware, Online Available: `http://www.gamemiddleware.org/`, Nov. 2010.

[9] HOLOBLOC, Online Available: `http://www.holobloc.com/`, Oct. 2010.

[10] A. Huhmann, *SPS/IPC/DRIVES 2004*. Berlin: FRANZIS, 2004, vol. Elektrische Automatisierung, Tagungsband, ch. Ethernet Networks for industrial Applications, pp. 107–115.

[11] IRRLICHT API, Online Available: `http://irrlicht.sourceforge.net/docu/index.html`, Sep. 2010.

[12] IRRLICHT Homepage, Online Available: `http://irrlicht.sourceforge.net/features.html`, Sep. 2010.

[13] Jani Peusaari, Jouni Ikonen, and Jari Porras, "Distribution issues in real-time interactive simulation," 2009 13th IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications, pp. 227–230, 2009.

[14] Lars Bishop and et al, "Designing a pc game engine," IEEE Volume 18, Issue 1, Jan./Feb. 1998, pp. 46–53, 1998.

[15] LDraw, Online Available:
`http://www.ldraw.org/`, Oct. 2010.

[16] LDraw2lws, Online Available:
`http://www.ldraw.org/download/win/ldraw2lws.shtml`, Nov. 2010.

[17] LEGO Homepage, Online Available:
`http://mindstorms.lego.com/en-us/default.aspx`, Sep. 2010.

[18] LEGO Motor Characteristics, Online Available:
`http://www.philohome.com/nxtmotor/nxtmotor.htm`, Sep. 2010.

[19] Lightwave, Online Available:
`http://www.newtek.com/lightwave/`, Nov. 2010.

[20] MS VisualStudio 2005, Online Available:
`http://www.microsoft.com/germany/VisualStudio/`, Nov. 2010.

[21] $\mu$Crons, Online Available:
`http://www.microns.org/`, Nov. 2010.

[22] NXTAPI, Online Available:
`http://lejos.sourceforge.net/p_technologies/nxt/nxj/api/index.html`, 2010.

[23] OGRE Homepage, Online Available:
`http://www.ogre3d.org/`, Sep. 2010.

[24] H. Peierl and W. Schlögl, *SPS/IPC/DRIVES 2004*. Berlin: FRANZIS, 2004, vol. Elektrische Automatisierung, Tagungsband, ch. Virtuelle Inbetriebnahme mechanisierter Fertigungszellen als Bestandteil der Digitalen Fabrik, pp. 123–131.

[25] Richard M. Fujimoto, "Parallel and distributed simulation," Proceedings of the 1999 Winter Simulation Conference, pp. 122+, 1999.

[26] Robert Lewis, *Modelling control systems using IEC 61499*. IEE Control Series, 59, 2001.

[27] H. Schludermann, T. Kirchmair, and M. Vorderwinkler, "Soft-commissioning: hardware-in-the-loop-based verification of controller software," Proceedings of the 32nd conference on Winter simulation, pp. 893–899, 2000.

[28] C. Szypersk, *WCOP'96 Summary in ECOOP'96*. dpunkt Verlag, 1996.

[29] Valeriy Vyatkin, *IEC 61499 Function Blocks for Embedded and Distributed Control System Design*. OOOneida, 2007.

[30] Windows Sockets 2, Online Available: `http://msdn.microsoft.com/de-de/library/ms740673.aspx`, Oct. 2010.

[31] Wolfgang Kühn, "Digital factory - simulation enhancing the product and production engineering process," Proceedings of the 2008 Winter Simulation Conference, pp. 1899–1906, 2006.

[32] M. Zyda, "From visual simulation to virtual reality to games," IEEE Computer, pp. 25–32, 1998.