TECHNISCHE UNIVERSITÄT DORTMUND

Faculty of Mechanical Engineering

IT in Production and Logistics (itpl)

Institute for Research and Transfer, RIF e.V.

Automation and Robotics

Master Thesis

Development of Generic PLC Integration Concept for a Graphical Simulation System

Univ.-Prof. Dr.-Ing. Markus Rabe

Univ.-Prof. Dr.-Ing. Jürgen Roßmann

Submitted by: Shilpa Appannavar

Matriculation number: 163185

September 2015

Eidesstattliche Versicherung

_____ _____

Name, Vorname Matr.-Nr.

Ich versichere hiermit an Eides statt, dass ich die vorliegende Bachelorarbeit/Masterarbeit*

mit dem Titel

_____

_____

_____

_____

selbstständig und ohne unzulässige fremde Hilfe erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate kenntlich gemacht. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

_____ _____

Ort, Datum Unterschrift

*Nichtzutreffendes bitte streichen

Belehrung:

Wer vorsätzlich gegen eine die Täuschung über Prüfungsleistungen betreffende Regelung einer Hochschulprüfungsordnung verstößt, handelt ordnungswidrig. Die Ordnungswidrigkeit kann mit einer Geldbuße von bis zu 50.000,00 € geahndet werden. Zuständige Verwaltungsbehörde für die Verfolgung und Ahndung von Ordnungswidrigkeiten ist der Kanzler/die Kanzlerin der Technischen Universität Dortmund. Im Falle eines mehrfachen oder sonstigen schwerwiegenden
Täuschungsversuches kann der Prüfling zudem exmatrikuliert werden. (§ 63 Abs. 5 Hochschulgesetz - HG -)

Die Abgabe einer falschen Versicherung an Eides statt wird mit Freiheitsstrafe bis zu 3 Jahren oder mit Geldstrafe bestraft.

Die Technische Universität Dortmund wird gfls. elektronische Vergleichswerkzeuge (wie z.B. die Software „turnitin“) zur Überprüfung von Ordnungswidrigkeiten in Prüfungsverfahren nutzen.

Die oben stehende Belehrung habe ich zur Kenntnis genommen:

_____ _____

Ort, Datum Unterschrift

Table of Contents

List of figures

List of tables

Abbreviations

PLC: Programmable Logic Controller

COM: Component Object Model

OPC: OLE (Object Linking and Embedding) for Process Control

SIL: Software In Loop

HIL: Hardware In Loop

RIL: Reality In Loop

1 Introduction

The frequent introduction of new and advanced products and product variants in the market significantly shortens the lifecycle of existing products. As the product lifecycles are reduced in the continuously changing marketplace, modern manufacturing systems should have sufficient responsiveness to adapt their behaviors efficiently to a wide range of circumstances. In order to remain competitive in today's market, the manufacturer should focus on improving the production system continuously alongside the improvements in the product itself [1]. The present day manufacturing systems need to be fast and flexible in terms of production, ramp up time and cost effectiveness.

Automation of the manufacturing process increases the product quality and quantity in comparison with the manual operations. Modern production lines are highly integrated systems, consisting of automated work stations such as robots with tool changing capabilities, a storage and hardware handling system, and a computer control system that controls the operation of the entire production line [2]. Involving many complex and interacting systems, commissioning of the production lines is an expensive and time consuming task. Decisions regarding the flow of material and placement of necessary equipment should be made very carefully during the design phase. A poor design can lead to large volume of inventory, inefficiency and inflexibility. Changing the layout in the later stages after the installation of all the machinery can be expensive. Thus it is necessary to get the design right to achieve the intended benefits.

Simulation helps to analyze complex manufacturing systems from the design and ongoing operation prospective. Technological developments and increasingly efficient computing platforms have significantly enhanced the process of modeling, simulation and analysis. This has enabled digital processing of complex tasks in manufacturing processes with the aid of high-fidelity models of the manufacturing system and the surrounding environment. The use of simulation models in the design and planning of manufacturing systems help to identify bottlenecks and detect the errors in scheduling of the processes. Some of the simulation tools used in academics and industries for example, ARENA and Automod are suitable only for the abstract design stage of the

production line. They do not include the behavior of the control systems into the simulation. These models are not suitable for a detailed design and analysis [2].

Production lines are usually controlled by different types of controllers for example, microcontrollers and Programmable Logic Controllers (PLCs). For more complex applications, PLCs are largely preferred over microcontrollers. Unlike microcontrollers, PLCs can be reprogrammed any number of times and also have a larger flexibility of adding more inputs and outputs. This serves as an advantage for PLCs over microcontrollers. PLCs are the most widely accepted controllers in the modern manufacturing industries. The simulation tools mentioned earlier lack this detailed level of control logic which regulates the material flow between different processes.

A detailed simulation model enables virtual verification of the production line which can foretell the production capability of the system and also verify the correctness of the control programs in conjunction with the surrounding equipment [2]. As PLCs are the most preferred type of controllers in the industries, integrating PLCs into the graphical simulation environment helps in generating a more realistic model. Verifying the PLC programs with the 3D simulation models enables detection of errors in the early phase. Correcting the errors in the control program in the early phase reduces the actual commissioning time of a production line by up to 25% [3].

The objective of this master thesis is to develop a generic PLC integration concept for a graphical simulation system, such that simulation models can be verified along with the control programs. Developing a controller object in the simulation environment which can interpret the PLC programs is a tedious task. Instead, external PLCs can be used to control the behavior of the simulation model. External PLCs can reproduce a more accurate behavior of the control system. More specifically, external hardware PLCs can be used to carry out the verification process under laboratory conditions.

To get a good overview of the thesis, it is very important to get an introduction into the important knowledge areas like virtual commissioning, PLCs, simulation software and different modes of communication. The state of the art chapter gives an overview of actual commissioning and concepts underlying the virtual commissioning. Further, this chapter introduces to the different software and tools used to realize the concept of generic PLC integration.

In this master thesis a generic controller object is developed in the simulation environment which communicates with external PLCs. The external PLC can be a simulated or hardware PLC from any PLC manufacturer. The controller object receives the input signals from various sensors in the simulation environment and sends these input signals to the external PLC. The external PLC then generates appropriate output signals after processing the input signals and the user program stored in the memory of the PLC. These outputs are written back to the outputs of simulation environments controller object. The controller object's outputs are connected to the actuators in the simulation environment. The Methodology chapter provides an overview of the controller object in the simulation environment and explains its generic nature and ability to connect to more than one external PLC.

Adhering to the concept demonstrated in the methodology chapter, a generic controller object is developed. The Implementation chapter explains the development and features of the generic controller object. To demonstrate the ability of the generic controller object to connect to different PLCs, connections to two types of external PLCs, a simulated PLC and a hardware PLC are implemented in this master thesis. Connections to both these external PLCs can be configured in the controller object. Communication between the controller object and the simulated PLC is via a COM object and the communication between the controller object and the hardware PLC is via an OPC client-server configuration.

The Validation chapter demonstrates the operation of the controller object with the help of a mechatronic model developed in the VEROSIM simulation environment. Inputs and outputs of the controller object are connected to the sensors and actuators in the model. PLC programs to control the behavior of the mechatronic model are developed using PLC program development software and downloaded on to the simulated PLC and hardware PLC. Connections to both simulated PLC and hardware PLC are configured in the controller object. The behavior of the mechatronic model is validated by making one of the configured connections active at a time. The response of the PLC program and the mechatronic model is validated by switching between the configured connections to the two external PLCs.

2 State of the Art

Virtual verification of the production lines can be done if 3D simulation models of the production line are more realistic. Integrating PLCs into the simulation environment makes the models more realistic and the mechatronic behavior of the production line can be verified in conjunction with control programs. This helps in reducing the errors detected in the actual commissioning phase, thus reducing the unnecessary delays and additional costs. This chapter includes a detailed description of the concepts underlying virtual commissioning and its advantage over traditional commissioning without prior virtual simulation. After understanding the concept of virtual commissioning, the requirements for realization of a virtual commissioning project are listed. Further different tools and software used to demonstrate the PLC integration concept is this thesis are discussed.

2.1 Actual Commissioning

Traditional development of manufacturing systems is carried out in different phases: Design of production facility, mechanical engineering, electrical engineering and automation engineering [4]. The design of the production facility is the first step in the traditional commissioning process. It involves decisions concerning the layout of the factory floor, placement of the machines and the staff in the manufacturing operation. This is an important component in the overall operation in terms of manufacturing process effectiveness and meeting the needs of the employees. The basic objective of the facility design is to ensure smooth flow of materials and information through the system. A poor design can lead to large volume of inventory, inefficiency and inflexibility. Also changing the layout in the later stages after the installation of all the machinery can be very expensive, thus it is important to get the design right before the installation of the machines.

The second step is the mechanical engineering phase, which involves commissioning of all the heavy machineries, robots, sensors and actuators required for the manufacturing process. The placement of these components is done according to the plan developed in the previous step of facility design. The third step is the electrical engineering phase,

where the electrical energy supply is established to the various machines, sensors and actuators.

The next step is automation engineering. In this step various controllers required for controlling the machines are installed. The connections to the various equipment, sensors and actuators from the controllers are also established for example via a field bus. The different kinds of controllers used in industry are for example, microcontrollers, PLCs etc. Since the modern production lines are highly automated with complex tasks, the PLCs are the more obvious choice for the controllers. At this stage after installing the controllers the functionality of each device is verified. Any errors encountered are corrected. Once all the machines are running error free, the production of the goods begins.

All the steps mentioned in traditional method of commissioning are carried out in a sequential manner. The verification and documentation of each of the stage is also performed separately [5]. Following the traditional method of commissioning, installation of PLCs and verification of the PLC programs is done in the last stage of commissioning. Thus the automation engineer has to wait for the verification of the PLC programs till all the machines and the surrounding equipment are installed in the production site. Since the PLC programs are not verified in conjunction with the surrounding machinery before installation, errors may be encountered, which in some cases can cause damage to the surrounding machines or the PLC itself or both. This leads to unnecessary delays and additional costs.

The modern production plants are highly automated and a majority of the functionalities are controlled by the control programs in these plants. As cited by Reinhart and Wünsch [3], up to one fourth of the total project life cycle time is accounted by the process of commissioning. Delays and activities related to electric and control devices consume up to 90% of the total commissioning time. Out of this nearly 70% of the delays are caused because of the errors in the control software. This is illustrated in figure 2.1.

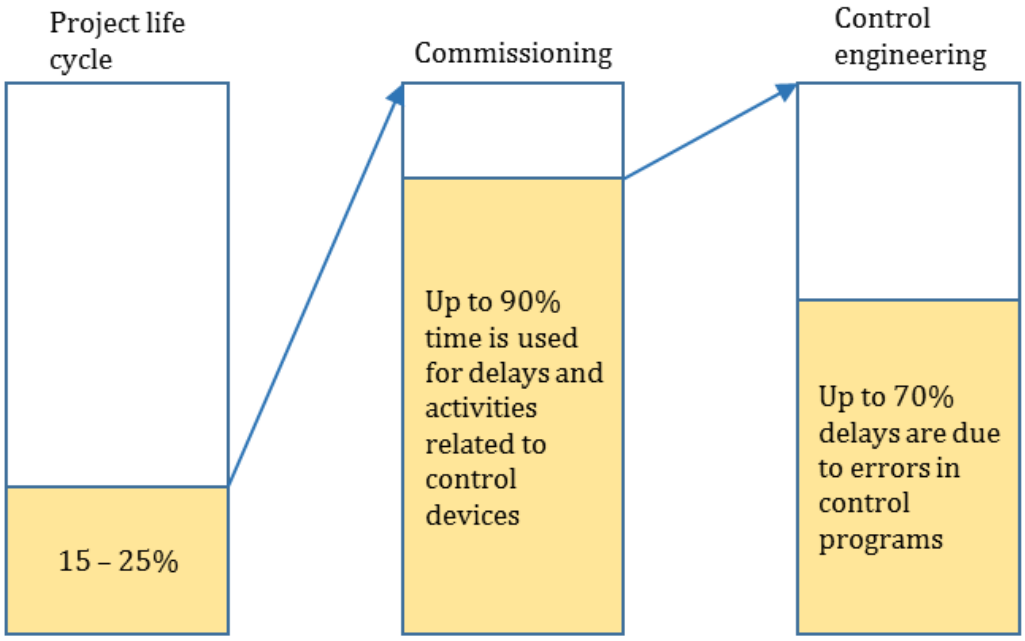


Figure 2.1: Delays in the process of commissioning [3]

The delays caused due to the errors in the control program can be minimized by verifying the correctness of control programs and its interaction with the surrounding machines. Using a virtual environment for the verification is a safe and economic solution.

Simulation is a powerful tool to overcome the hurdles witnessed in the traditional method of commissioning to a larger extent. It helps to analyze the manufacturing systems from the design and ongoing operation prospective. Technological developments and increasingly efficient computing platforms have significantly enhanced the process of modeling, simulation and analysis. This has enabled digital processing of complex tasks in manufacturing process with the aid of high-fidelity models of the manufacturing system and the surrounding environment. 3D simulation models help in validation of the systems prior to installation, thus accelerating the process of planning and implementation of manufacturing systems. As indicated in [6], realization of the production system in terms of 3D models belong to the scope of *Digital Factory*. Digital factory is defined as "a generic term for a comprehensive network of digital models and methods including simulation and 3D visualization. Its purpose is the integrated planning, implementation, control and continual improvement of all essential factory processes and resources associated with the product" [7].

2.2 Virtual Commissioning

Use of realistic and accurate 3D simulation to validate the functions of production equipment control system prior to actual implementation is termed as "Virtual Commissioning" [8]. Virtual Commissioning can be a valuable tool for assembly line design engineers in a sense that it can provide decision making support to important decisions like for example, type and number of resources to be used within an assembly line or selection of communication and interfacing protocols between the resources [4].

Virtual commissioning serves as an advantage during the actual commissioning process by minimizing the delays and additional costs. Tests conducted to evaluate the cost benefits of virtual commissioning as described in [3], shows that the software quality in terms of fulfilled requirements was improved by more than 100% whereas the commissioning time was reduced by 25%.

In the process of Virtual Commissioning a virtual model of the plant is used to replicate the mechatronic behavior of the different machines and equipment and a real or simulated PLC is used for the control programs. The possible combinations between reality and simulation with the aim of verification of the PLC programs are as shown in the figure 2.2.

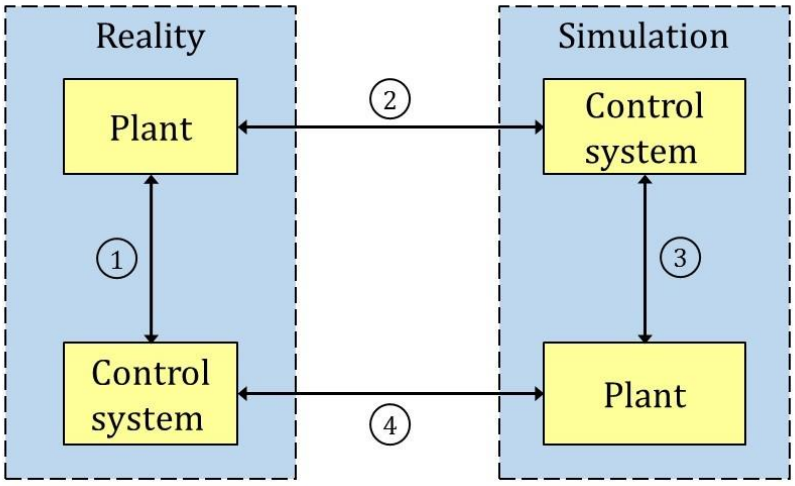


Figure 2.2: Possible combinations between reality and simulation [9]

1 **Traditional commissioning:** real plant and a real control system.
2 **Reality in Loop (RIL):** real plant and a simulated control system.
3 **Software in Loop (SIL):** Simulated plant and a simulated control system.
4 **Hardware in Loop (HIL):** simulated plant and a real control system.

Involving a real plant and a real control system (1) is the traditional method of commissioning. Any errors encountered in the control programs might cause serious damage to the plant equipment or PLC or both. This leads to additional delays and costs during the commissioning process. As indicated earlier a major part of the control engineering during commissioning is accounted to the errors in the control programs.

The second combination is RIL (2) method. This includes a real plant and a simulated PLC. Since a real plant is involved in this method, the risk of damaging the machines and equipment in the plant is still not eliminated.

The SIL (3) method of virtual commissioning is one of the safest and economic methods. This includes a simulated plant and a simulated control system, therefore minimizing the costs caused by the errors in the control program. This method of verification can be performed parallel to the earlier steps of actual commissioning once the design of the production facility is done. Thus SIL method of virtual commissioning saves a lot of time during the actual commissioning. One of the disadvantages of SIL method as pointed out in [3] is, the low availability of up-to date control simulation packages for a particular control version. Therefore the control software cannot provide an exact reproduction of the control behavior [4].

In the last method of HIL (4) simulation, a virtual plant model is used in conjunction with a real control system. In this method the advantage over the SIL method is that the control engineers can perform the testing of complex control and automation scenarios under laboratory conditions [3]. Also the PLCs used for verification of the control programs along with the virtual environment can be directly used in the real production facility.

Both SIL and HIL are less costly approaches compared to the rest, as virtual environments are used instead of real world systems. These approaches minimize the errors in the PLC programs, reduce the risk of damaged equipment subsequently

decreasing the actual commissioning time and increasing the quality of manufacturing system.

2.3 Requirements for Realization of a Virtual Commissioning Project

The following are the requirements for the realization of a virtual commissioning project [4]:

1 3D models of the equipment and other resources to be commissioned, including information about the geometries, kinematics and electrics.
2 The layout of the production facility, involving the placement of all the resources and equipment.
3 Knowledge of the material flow in the production facility, including the sequence of operations and interdependencies in the production process.
4 The real or a simulated control system. Either a real PLC or a simulated one can be used to demonstrate the two possible methods for verifying PLC programs.
5 Detailed definitions of the PLC's input and output (I/O) signals including their respective mapping to the resources within the simulated environment.
6 Detailed definition of the extra functionalities and signals such as, alarm or safety systems that are to be included in the commissioning process.
7 Software drivers and communication protocols for establishing the communication between the simulation model and the control system.

2.4 Simulation Software

Modeling and simulation software are of great help in the field of engineering. It is very common to use block oriented simulation systems such as MATLAB/SIMULINK [10] or Modelica modeling language [11] to develop controllers . FEM analysis tool (for example, cosmol [12]) are used to test and verify component layouts. Increasingly more efficient computing platforms have facilitated the development for 3D modeling and simulation software which are capable of producing more realistic replica of the real system. Focusing on 3D simulation technology, various approaches can be found for development of mobile robots (for example, gazebo [14]) or generic mechatronic systems

(for example, simmechanics from mathworks [16]). As stated in [13] these software still lack a holistic and encompassing approach that enables and encourages the synergetic use of simulation methods on a single database throughout the entire lifecycles of technical systems. A new simulation system architecture was developed to overcome these limitations.

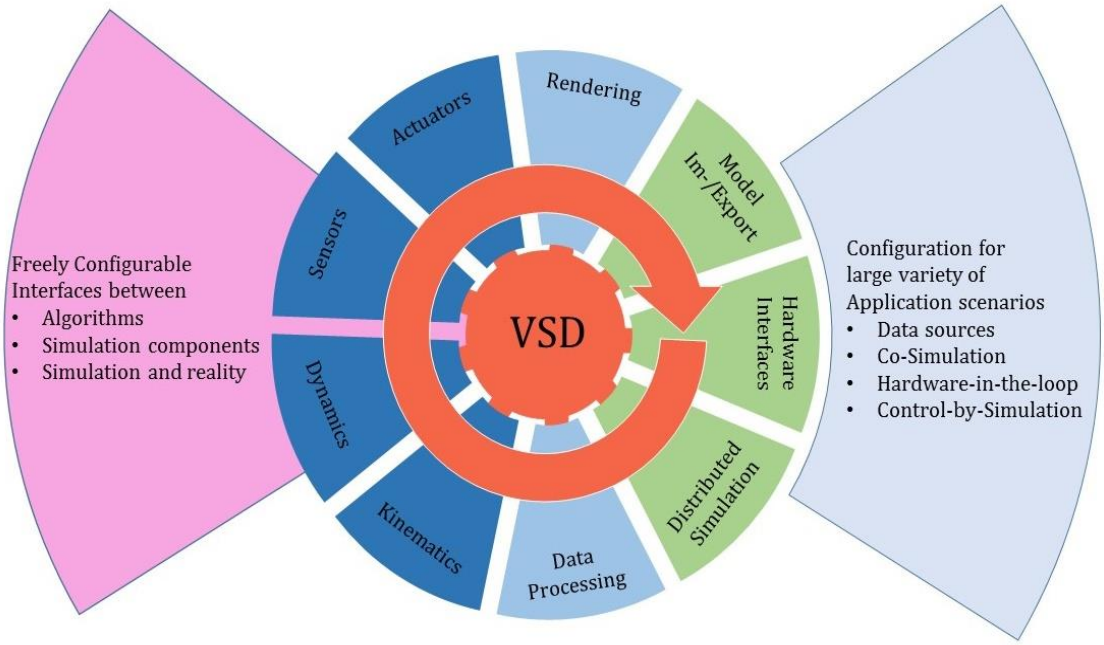


Figure 2.3: VEROSIM micro kernel architecture [13]

The idea of VEROSIM is introduction of a micro kernel, the "Versatile Simulation Database" (VSD) figure 2.3. VSD is an object oriented real-time database holding a description of underlying simulation model. It is fully implemented in C++ and provides the central building blocks for data management, meta information, communication, persistence, and user interaction. VSD is said to be "active" as it contains algorithms and interfaces to manipulate data along with static data container. Furthermore VSD also indicates the user about addition, deletion or change in event through intelligent messaging systems. In addition VSD also provides essential functionalities for parallel and distributed simulation. Specialized plugins are built upon the VSD core that aids in achieving all simulation functionality of the framework by interacting with the VSD core.

The comprehensive 3D simulation system VEROSIM is built on the basis of the basic simulation system architecture. The key modules like for example data storage,

visualization, kinematics, sensor/actuator framework and more for 3D simulation are developed by extending the VSD kernel in various directions.

All the key components are accessible via a comprehensive graphical user interface in desktop applications and projection environments. A textual interface is available on platforms which do not support 3D rendering. Also all these interfaces can be used simultaneously.

The VEROSIM 3D simulation framework has been used to realize a variety of different applications, currently focusing more on the areas of environment (e.g. forest inventory or forest machines), industry (e.g. industrial automation) and space (e.g. space robots).

The figures 2.4 to 2.6 show the user interface of the VEROSIM environment. Figure 2.4 is the main window for modeling the mechatronic systems. The ‘Model explorer” (on the left side) holds the list of all the elements added to the model by the user. The ‘Property widget’ (on the right side) shows the properties of the element selected element in the model explorer. For example the property ‘Modify frame’ allows the user to modify the position and the property ‘Scaling’ allows the user to modify the size of the selected element.

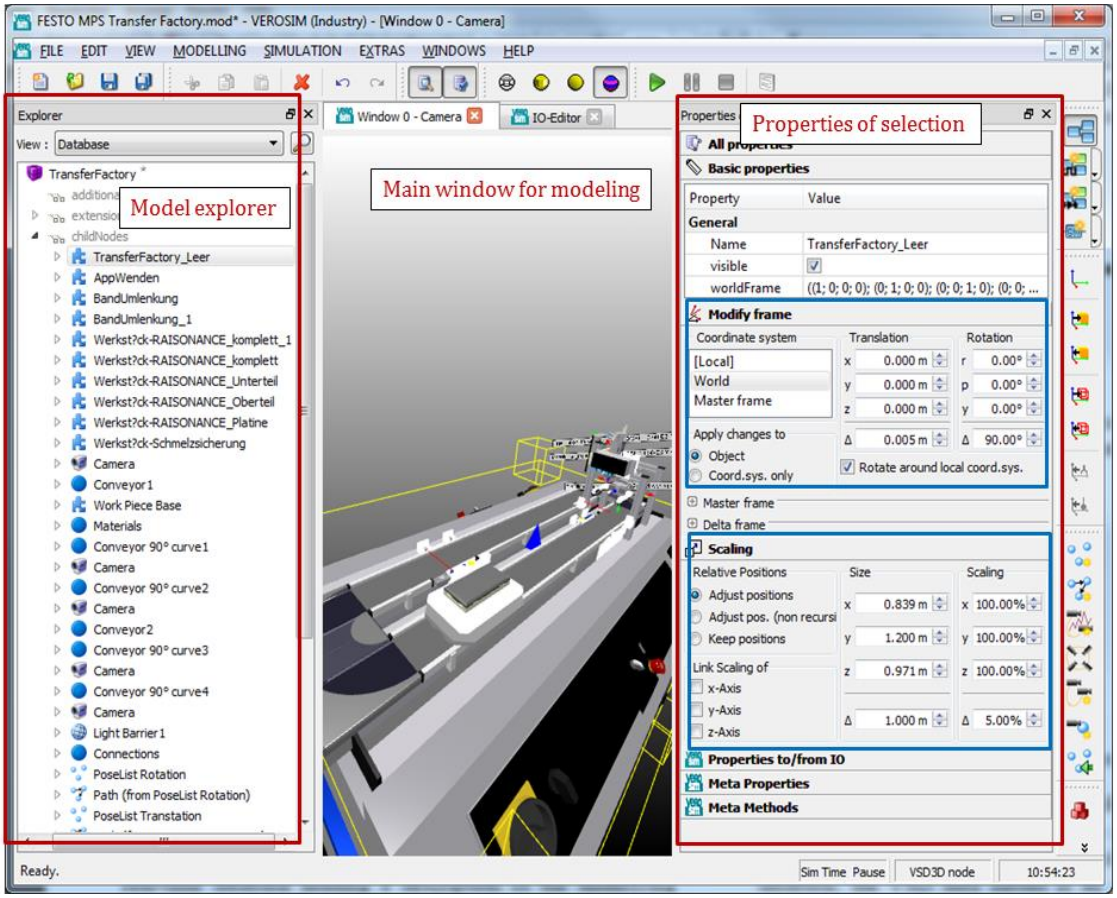


Figure 2.4: VEROSIM modeling environment

The IO editor (figure 2.5) enables the user to make the connections between various components.

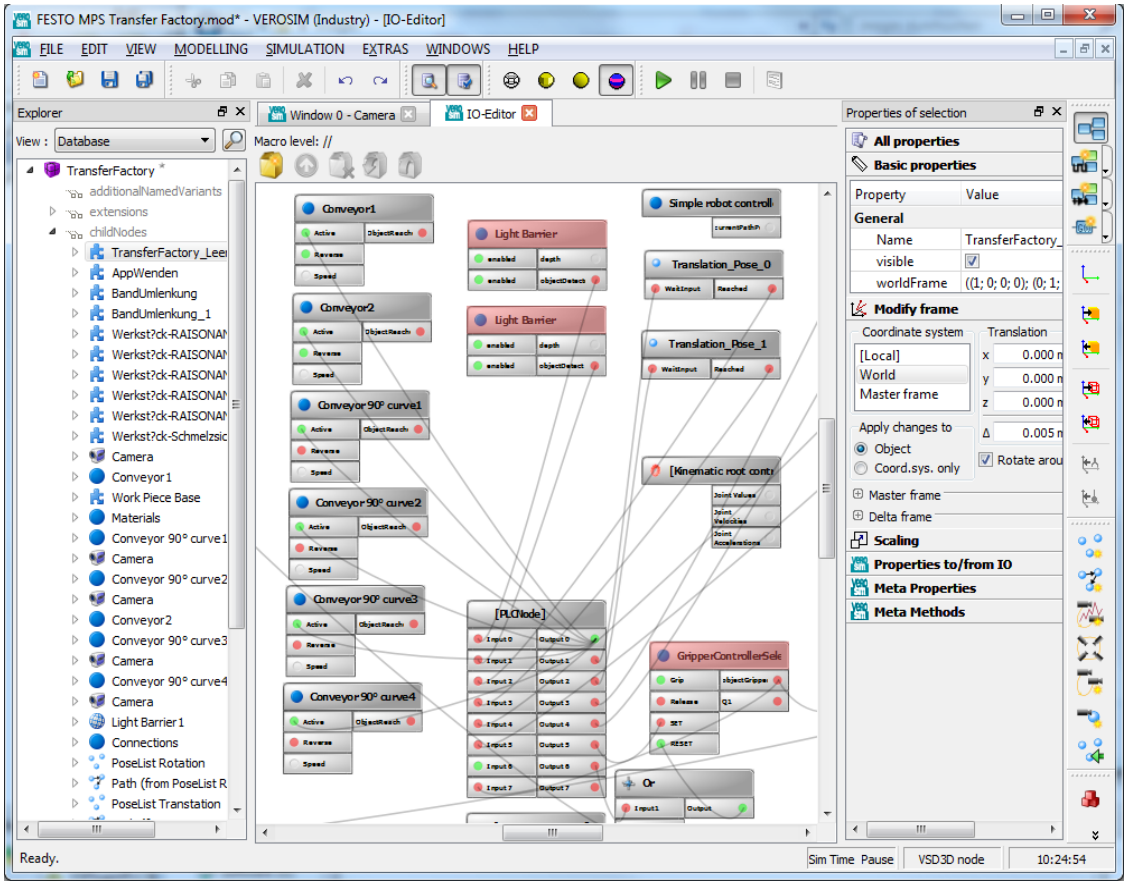


Figure 2.5: VEROSIM IO editor

The model library (figure 2.6) gives a list of industrial components and robots offered by VEROSIM. These components can be readily used by the user in developing complex systems.

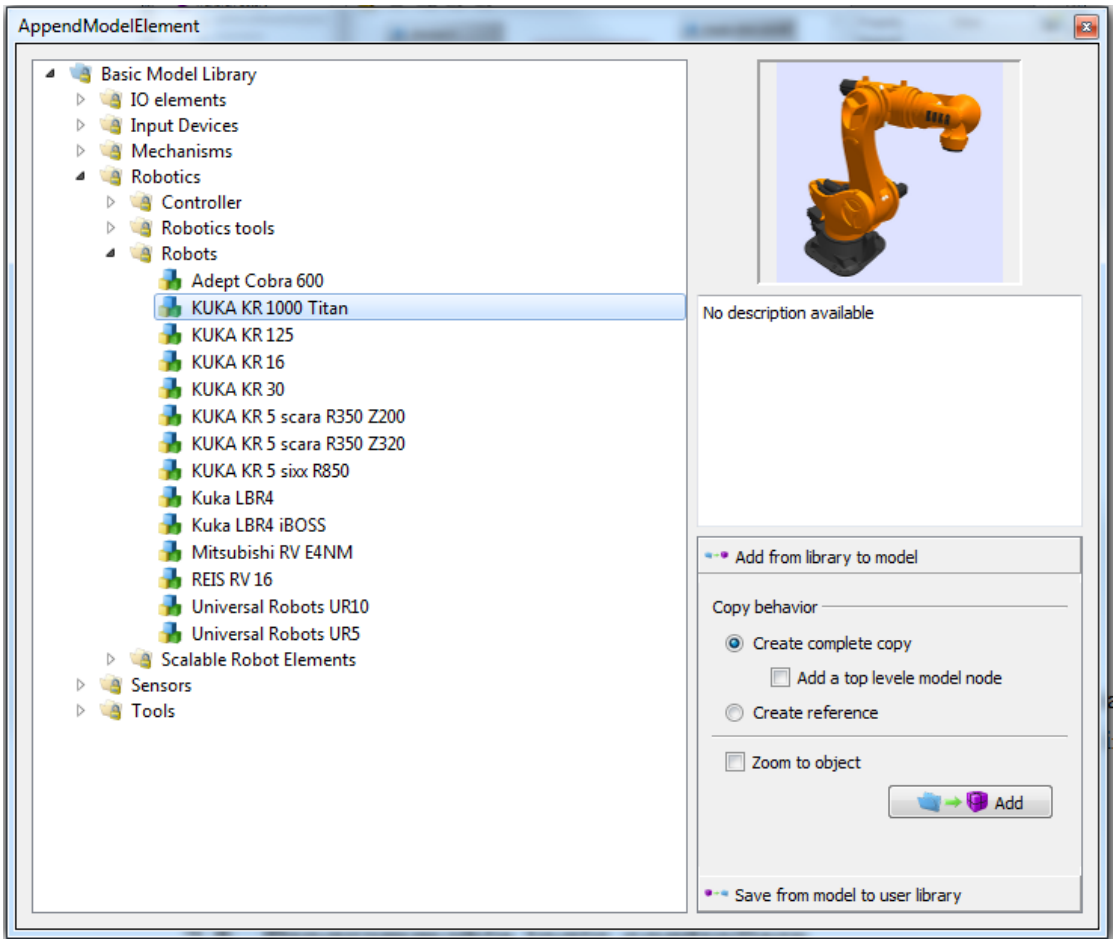


Figure 2.6: VEROSIM model library

2.5 Programmable Logic Controllers

"A programmable logic controller (PLC) is a special form of microprocessor-based controller, that uses a programmable memory to store instructions and to implement functions such as logic, sequencing, timing, counting and arithmetic in order to control machines and processes" [15]. Having the ability to be re-programmed any number of times and able to control more than one device at a time makes PLCs a more preferred choice over other controllers like for example, microcontrollers.

PLCs are similar to computers but built in a more suitable way for industrial environments. They are designed in a rugged way to withstand the noise, vibrations, temperatures and humidity in an industrial environment. Components of a typical PLC system are as shown in the figure 2.7.

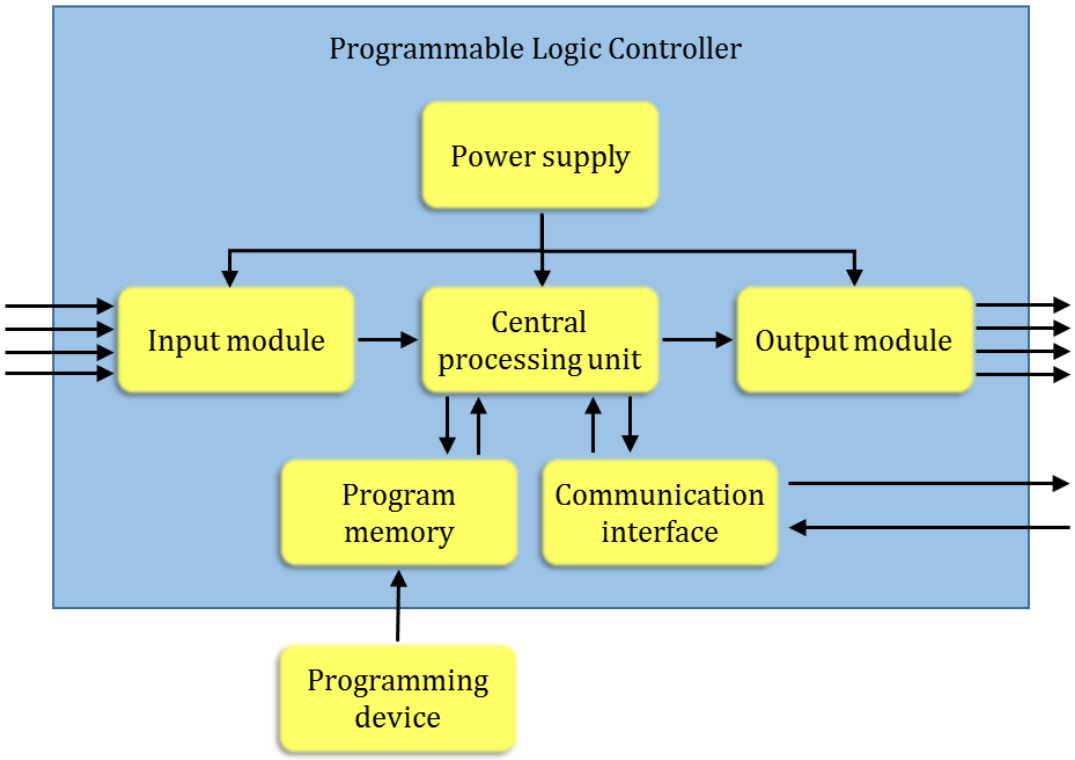


Figure 2.7: PLC system [15]

- The *power supply* unit converts the incoming high voltage power to the required voltage levels for processor, circuits in the input and output module and other components.
- The *Central Processing Unit (CPU)* generates the output signals for the control actions depending on the inputs received and the instructions stored in the program memory of the controller.
- The *programming device* is used to develop the PLC programs, which are then downloaded on to the memory of the PLC device.
- The *memory* of the PLC system consists of several elements such as, ROM for permanent storage of the operating system and the data used by the system, RAM for the storage of user programs and data.

- *Input module* is where the device receives the input signals from various sensors and switches.
- The *Output module* communicates the signals to the external actuators like for example, motor, valves etc. The input and output modules can be digital or analog depending on the type of sensors and actuators connected.
- Transmitting and receiving data from remote PLCs, Human Machine Interface (HMI) panels, SCADA (Supervisory Control And Data Acquisition) systems etc. are handled by the *Communication interface*.
- The *Programming device* can be for example: a personal computer. The PLC manufacturers have programming software for their PLCs. For some PLCs a special communication card is required to download the programs onto a real PLC whereas for some PLCs just having the appropriate software on the computer is sufficient.

The two common types of mechanical design for PLC system are, a single box type or a modular/rack type. The single box type is supplied as an integral compact package with power supply, processor, memory, inputs and outputs. Since the number of inputs and outputs are fixed in this type of PLC, they are generally preferred for small scale applications. The PLCs of modular/rack type are more flexible in terms of number of inputs and outputs. This type of PLC system consists of separate modules for each of the PLC components like, power supply, CPU, inputs and outputs, communication interface, etc. The user can decide about the number and type of modules required depending on its application. Thus the number of input and output modules can be expanded depending on the application the PLC is used for. [15]

For programming, most of the PLCs adopt the programming languages described by the international standard IEC 61131-3. The IEC 61131-3 defines textual and graphical languages for writing the application programs [17]. The PLC manufacturer provides programming software which allows the user to write the application programs in any of the standard programming languages. For example, Siemens has the software *SIMATIC STEP 7*, which complies with the international standard IEC 61131-3. Following is a brief description of the standard programming languages as defined by IEC 61131-3:

- *Ladder logic (LD):* This is one of the most commonly used methods of programming the PLCs and this is equivalent to drawing a switching circuits. Two vertical lines

represent the power rails in the ladder diagram. And circuits are connected between these two vertical lines.

- *Function block diagram (FBD):* In FBD the PLC programs are described in terms of graphical blocks. A FBD is a program instruction unit with one or more inputs and yields one or more outputs when executed.
- *Instruction list (IL):* This is a textual programming language similar to the assembly programming language. The programs written in instruction list consists a series of instructions, each beginning on a new line. It consists of operators followed by one or more operands. Each instruction is represented by using mnemonic codes. The codes differ from one manufacturer to other.
- *Sequential Function Chart (SFC):* The SFC is used to represent the operation in a pictorial format. It shows the sequence of events involved in the operations. Different events are represented by steps or states. Each step is connected to the next one through transition conditions, which have to be realized before the execution can move on to the next step from the previous step.
- *Structures Text (ST):* The ST language contains list of statements separated by semicolon. The Siemens Structure Text language is known as SCL (Structure Control Language). It is a high level textual programming language based on PASCAL (S7 SCL manual referred).

As a general practice complex control tasks are broken down into many simpler tasks and each task is implemented in a separate block which is termed as *Program Organization Unit (POU)* [17]. The PLC executes these POU's continuously in a cyclic manner, reading the inputs and updating the outputs [15] (figure 2.8).

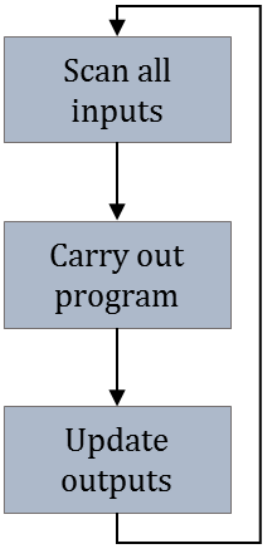


Figure 2.8: PLC operation [15]

In this master thesis, the PLC integration concept is demonstrated using a simulated PLC and a real hardware PLC are used. The simulated PLC *S7–PLCSIM* and the hardware PLC *S7-300* from Siemens are used. The Siemens software *STEP 7* is used to write the control programs for the model developed using VEROSIM. The Siemens STEP 7 (version) standard software package is used for configuration of the simulated and hardware PLC.

2.5.1 S7-PLCSIM

The main advantage of using a simulated PLC with the virtual model of production line is that the control programs can be verified without the use of any hardware. Since no hardware component is involved in the process of verification of the model, this eliminates the risk of damaging any of the machines or controllers involved in the production line. S7-PLCSIM is the simulated PLC made by Siemens, which allows the user to test the control programs without connecting to the S7 hardware.

S7-PLCSIM provides a simple graphical user interface to the STEP 7 user program, which can be used for monitoring and modifying different objects like for example, input and output variables. Also the operating mode of the simulated PLC can be controlled via the interface.

The simulation view window is as shown in the figure 2.9. The CPU is added to the simulation window automatically on start of the S7-PLCSIM application. Later the variables required to be monitored can be added from the option in the toolbar. [18]

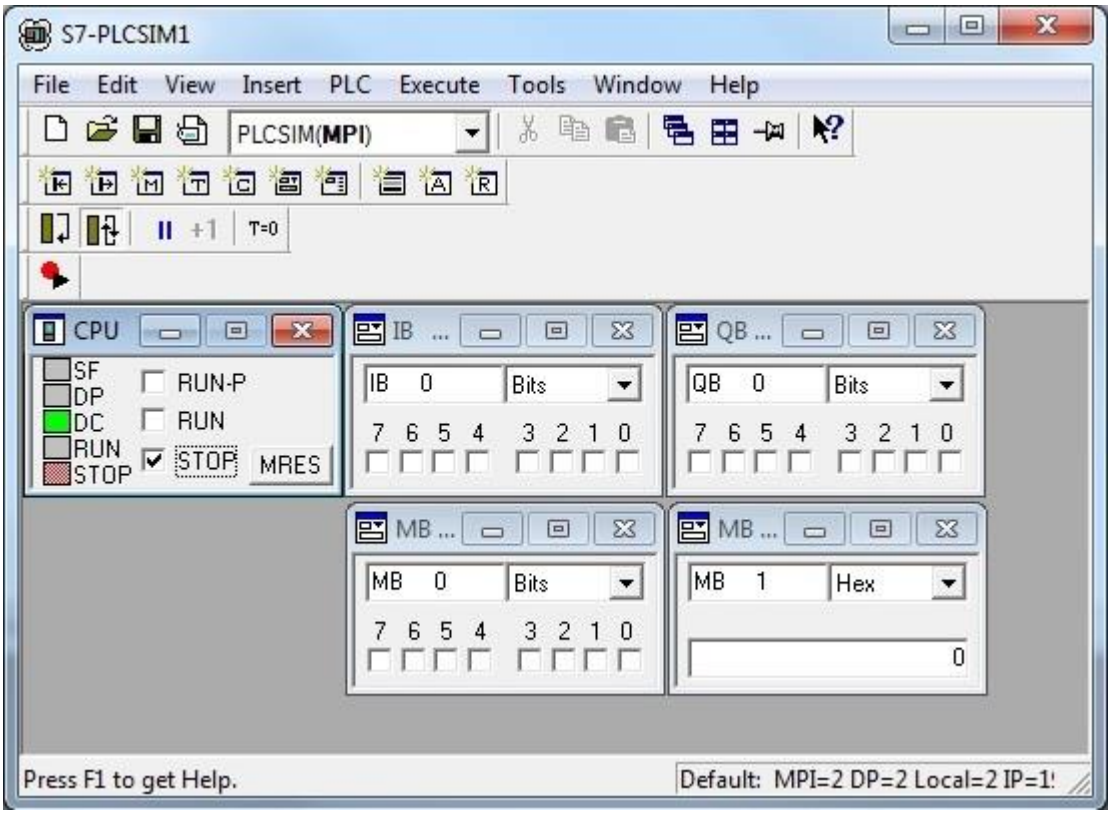


Figure 2.9: PLCSIM simulation view window

2.5.2 Siemens S7-300 PLC

The second type of PLC used to demonstrate the generic PLC integration concept is a real hardware PLC. As indicated earlier the low availability of up-to date control simulation packages for a particular control version serves as an disadvantage for using a simulated PLC [3]. Using a hardware PLC the control engineers can perform the tests under more realistic conditions.

A Siemens S7-300 PLC is used in this thesis. This is a modular/rack type PLC. The first module on the PLC rack is the power supply. Followed by the power supply is the CPU and the input and output module. Last is the communication module which helps in connecting the PLC with the PC.

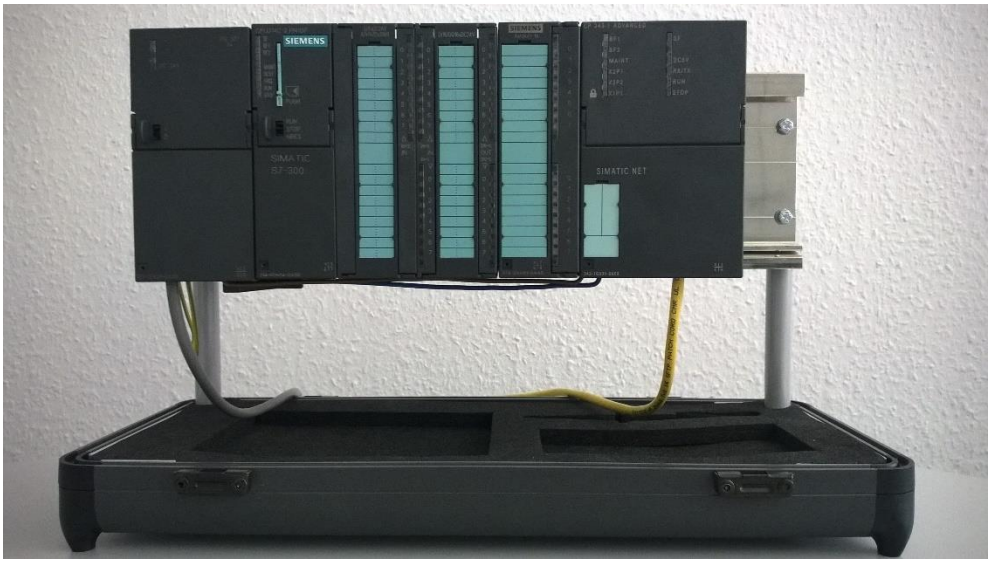


Figure 2.10: Siemens S7-300 PLC

The CPU used here is *CPU 314-C 2 PD/DP*. The input output module is DI8/DO8xDC24V/0.5A. The communication interface used is CP 343-1 Advanced. The communication interface communicates efficiently with the PLC's CPU, OPC server and the programming device via a TCP connection.

2.5.3 Siemens STEP 7

STEP 7 is the standard software package by Siemens used for the purpose of configuring and programming SIMATIC programmable logic controllers. The STEP 7 standard package provides a series of applications within the software. The figure 2.11 shows the applications/tools provided by the STEP 7 standard [19].

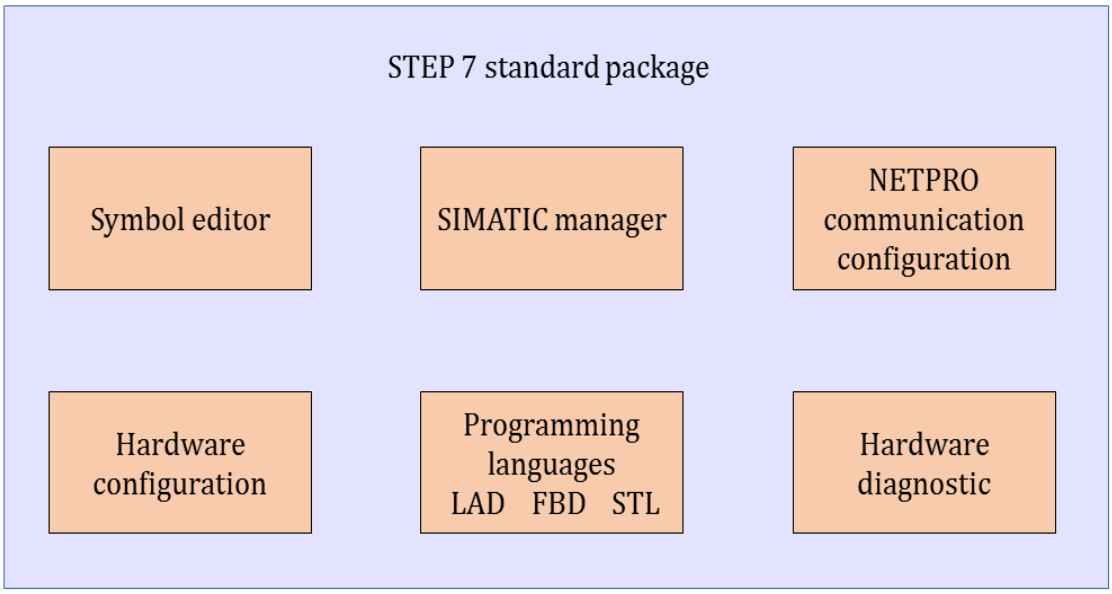


Figure 2.11: Siemens STEP 7 standard software package [19]

- *SIMATIC manager:* The SIMATIC manager manages all the data that belongs to the automation project.
- *Symbol editor:* The symbol editor manages all the shared symbols. This allows the user to set the symbolic names and comments for the process signals.
- *Programming languages:* The SIMATIC programming languages integrated in STEP 7 are compliant with IEC 61131-3 standard PLC programming languages. Ladder logic (LAD), Statement list (STL), and Function block diagram (FBD) are an integral part of the standard STEP 7 package. Other programming languages are available as optional packages.
- *Hardware configuration:* The hardware configuration tool enables user to configure and assign parameters to the hardware of an automation project.
- *Hardware diagnostic:* The hardware diagnostic provides an overview of the status of the PLC. This displays general information of the module and its status. Any faults in the modules are also indicated.
- *NetPro (Network configuration):* Using the NetPro cyclic data transfer via the MPI is possible, where the user can the connections and blocks to the CPU.

2.6 Communication Modes

In order to simulate the behavior of the virtual model according to the control programs on an external PLC, communication needs to be established between the virtual model and the external PLC. Since in this master thesis two different kinds of PLCs are used, both of them communicate with the virtual model using different modes of communication. For communicating with the simulated PLC S7-PLCSIM a COM object called S7ProSim is used. In case of the hardware PLC, the communication between the virtual model and hardware PLC is established though an OPC (OLE for Process Control) connection. (OLE: Object Linking and Embedding).

2.6.1 S7ProSim

S7Prosim is a COM object, which provides programmatic access to the simulated PLC S7-PLCSIM. Using S7ProSim the user can write software programs that can perform tasks such as running the control program in single scan mode, reading or writing controller values, and many other work steps.

To be able to use the S7ProSim COM object in the project, a reference is added to the object in the project. After adding the reference to the project, various methods provided by the COM object are used to communicate with the S7-PLCSIM. Following are a few methods provided by the S7ProSim:

- Connect: Connects the S7ProSim to S7-PLCSIM.
- ConnectExt: Connects the S7ProSim to the S7-PLCSIM instance with number InstanceNumber.
- Disconnect: Disconnects the S7ProSim form S7-PLCSIM.
- GetScanMode: Reports the operating mode of S7-PLCSIM.
- ReadDataBlockValue: Reads a particular bit, byte, word, or double word from the DB memory area of S7-PLCSIM.
- ReadOutputImage: Reads elements from the peripheral outputs (PQ memory area) of S7-PLCSIM.
- WriteDataBlockValue: Writes a particular bit, byte, word, or double word to the DB memory area of S7-PLCSIM.

- WriteInputImage: Writes elements to the peripheral inputs (PI memory area) of S7-PLCSIM, starting at the StartIndex of the data to which pData points.

2.6.2 OPC

OPC is a technical specification that defines a set of standard industrial software interfaces based upon Microsoft's OLE/COM technology [20]. The acronym OPC is derived from OLE (Object Linking and Embedding) for Process Control. [OPCFoundation.org].

Before the development of the OPC standard, industrial software applications that access the process data were restricted to the access techniques of the communication network of one manufacturer. For a particular hardware device, each software application must include a separate driver. This posed some problems in terms of having devices from different vendors for one particular application. The customer was bound to use all the required components of the control system from the same vendor. The standardized OPC interface now allows the user to have uniform access to communication networks of any vendor via the OPC interface. [21]

There are several specifications released by the OPC foundation for enabling data communication on the basis of a client-server architecture. The interface which allows the exchange of events and data between application, OPC server, OPC client and the device are defined by the OPC specification. The most common specifications are described below in brief [22]:

- *OPC Data Access specification (OPC-DA):* This was the first specification released by the OPC foundation [22]. This enables communication of process data between one or more data resources and clients [23]. The data resource can be located on the same computer as the client or the data resource can be a sensor or control and automation unit connected through a communication network.
- *OPC Alarm and Event specification (OPC-AE):* The OPC Alarm and Event specification defines an interface for servers and client for transmission and acknowledgement of the alarms and events in a structured manner [22]. Data from different sources like for example, PLC, and sensors can be received by the AE

server. The OPC-DA server and OPC-AE server can access data from the same data resource. The difference is that, OPC-DA provides continuous data stream and the OPC-AE server's data stream is triggered by the change in the event like for example, temperature crossing a certain threshold etc. [20]

- *OPC Historical Data Access Specification (OPC-HDA):* The OPC-HDA server provides the client, access to the historical data. The historical data can be presented in a simple row data manner or in the form of aggregated data which is the processed data [22].

In this master thesis the OPC client- server software from Softing [reference needed] is used. The OPC server fetches the data from the real S7-300 PLC connected via a TCP/IP connection. The OPC client then accesses the data from the OPC server, which is later communicated to the 3D simulated model.

2.6.3 Softing OPC Server Ethernet

The OPC server Ethernet from Softing enables the exchange of data between the field device (PLC) from a wide range of manufacturers and OPC client via Ethernet TCP/IP. The OPC client server interface provided by Softing is based on the OPC-DA specification provided by the OPC foundation. An OPC server allows one or more OPC clients to access process data via this interface.

The OPC-DA specification allows exchange of data between one or more OPC servers and client in a continuous manner. In the OPC-DA specification three hierarchical classes are defined: OPCServer, OPCGroup, and OPCItem. An object of class OPCServer represents manufacturer specific OPC server. The variables used by OPC server are structured in the object of class OPCGroup. With the OPCGroup the client is able to create a list of useful variables. An OPCItem is the actual process variable accessed by the client.

The NetCon OPC is the graphic interface for configuring and diagnosing the OPC server. The GUI connects with the OPC server core via TCP/IP connection. The OPC server from Softing has the name *INAT TCPIPH1 OPC Server*. Along with the OPC server, a test OPC client is also installed. The OPC client determines the OPC servers already

installed and offers to connect to them on the start of the OPC client. After connecting to the OPC server, a new OPCGroup can be created in the client. The process variables of interest are added as OPCItem in the OPCGroup.

3 Methodology

This chapter focuses on the methodology adopted to integrate different types of external PLCs into a graphical simulation environment and the virtual verification of the simulated mechatronic model. Further this chapter provides a basic idea of the generic controller object developed in this master thesis and the design of the communication architecture between the controller object and the external PLC.

3.1 Steps for Virtual Verification of the PLC Program

Figure 3.1 shows the steps involved in the process of verifying the PLC programs in conjunction with a simulation model.

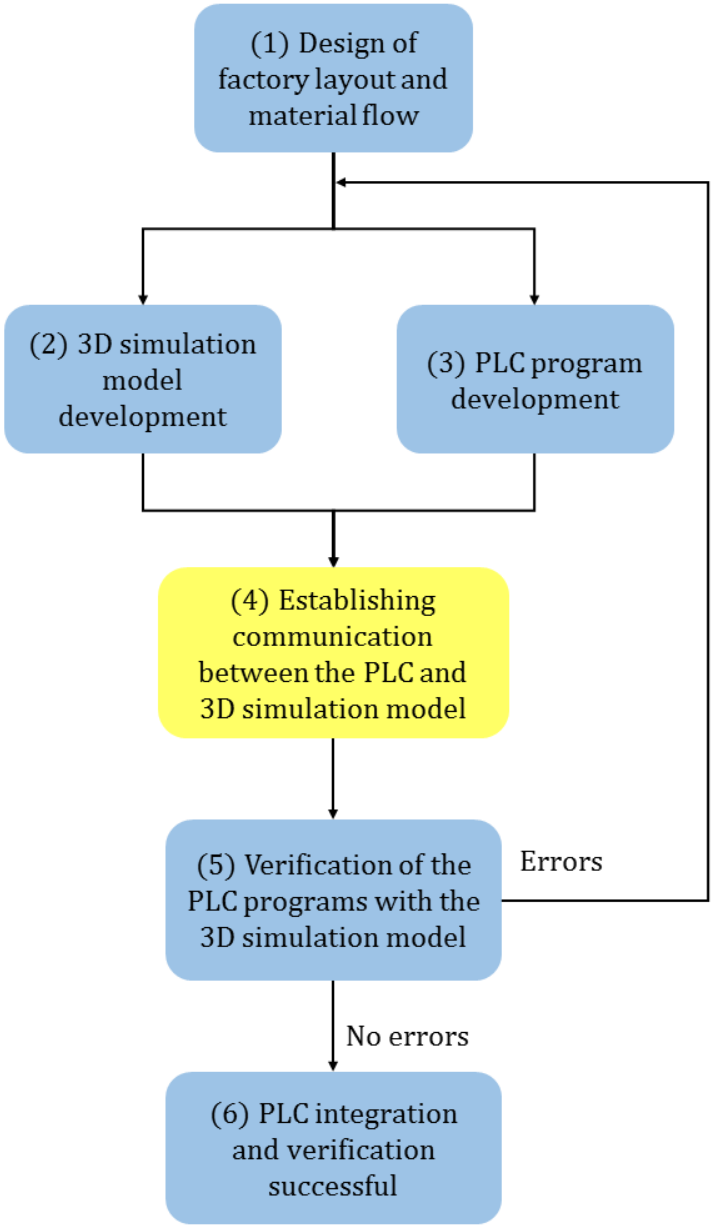


Figure 3.1: Steps in virtual verification of PLC programs

In step (1) the design for the factory layout is developed. This step defines the placement of various equipment, sensors and actuators, number of controllers required and the flow of material from one station to the next in the production system. As the flow of material from one station to the next in the production system is known, sequence of operations to be carried out by the controllers at every station can now be defined in brief. After the sequence of operations for each of the controllers is defined, the inputs and outputs of the respective controller can be listed. This list of inputs and outputs is used in the later stages of implementing the control programs.

In step (2) a 3D model of the production system is developed in accordance with the layout from the previous step using a simulation software. Some simulation software provide a library of basic industrial components such as conveyor belts, sensors, switches etc. These components can be directly used in developing complex systems. In case of absence of one or more components, CAD models of the missing components have to be developed by using other appropriate software and then import these CAD models into the 3D simulation environment.

Since PLCs are the more preferred choice of controllers in industries, the type of controllers used in this master thesis are PLCs. The process of implementation of the of the control programs which govern the behavior of the model can be done in parallel with the development of simulation model since, the material flow in the plant, sensors and actuators, and the list of inputs and outputs of the controllers in the simulation model are known in advance. As shown in the figure 3.1, step (3) represents the development of the PLC programs

After the PLC programs are implemented, they are downloaded to an external PLC. Communication should be established between the simulation model and this external PLC. This is shown as step (4) in the figure 3.1. The mode of communication between the simulation model and the PLC depends on whether the external PLC is a hardware or a simulated PLC, and on the possible means of communication offered by the PLC and the simulation software.

PLC programs can be verified with the model after a successful communication is established between the two (step (5) in the figure 3.1). The inputs from the sensors are transmitted to the external PLC, which generates appropriate control signals depending on the instructions stored by the user in the memory of the PLC. The outputs are communicated back to the simulated model. The models behavior should expose any errors in the control program, the model itself or both. Any errors in the simulated plant can be corrected in the simulation and in case these errors are also present in the design of the plant, the correction can be applied upstream to the design documents. The PLC programs can be edited with minimal effort and downloaded to the simulated or hardware PLC to correct any errors. Once the model and the PLC programs run error free, the control programs are ready to be downloaded to the real PLCs for the actual commissioning of the plant. If a simulated PLC was used for the simulation, an additional

simulation run with a hardware PLC might be included before commissioning to verify the correctness of the program on real hardware.

3.2 Communication with the External PLC

Figure 3.2 illustrates the basic idea adopted to integrate the external PLC into a graphic simulation environment. The simulation model consists of various objects like such as, work-pieces, sensors, actuators, conveyor systems etc. Communication between the simulation model and the external PLC is such that, the inputs from the sensors in the simulation model are connected to the inputs of the external PLC and the outputs of the PLC are connected to various actuators in the simulation model.

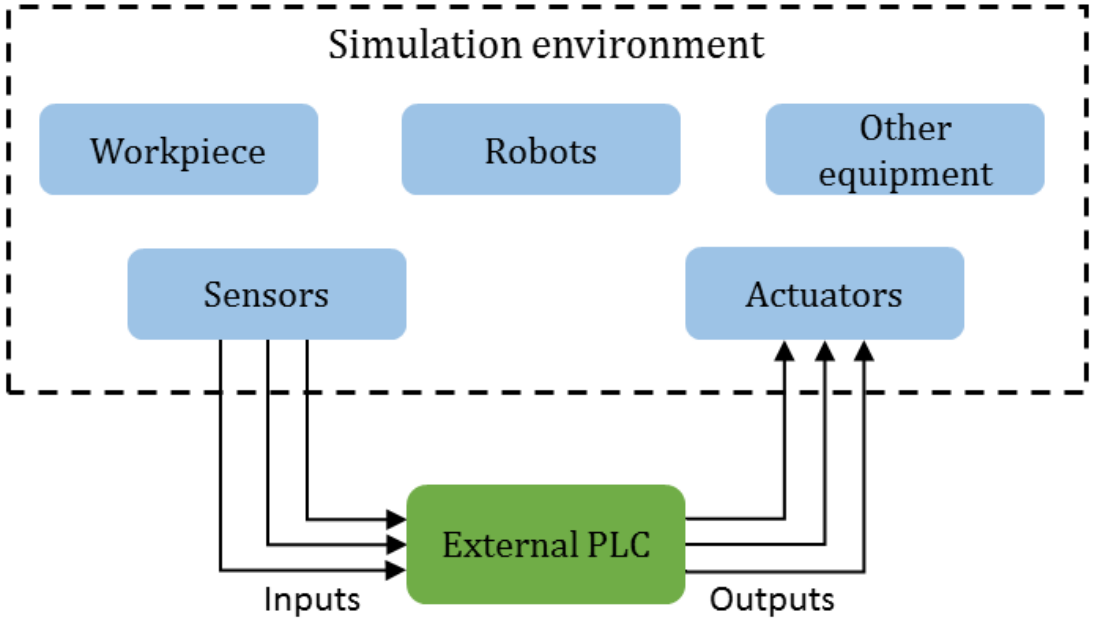


Figure 3.2: Integration of external PLC into a graphic simulation environment

For the simulation model to be able to exchange inputs and outputs with the external PLC an appropriate communication interface is needed. Figure 3.3 shows the concept of such an interface that facilitates the exchange of data between the simulated model and the external PLC.

Figure 3.3: Exchange of data between simulation model and external PLC

The choice of the communication interface between the simulation model and the external PLC depends on the software used for development of the simulation model as well as on the type of the external PLC. The sensor data from the simulation model is transmitted to inputs of the external PLC via the communication interface. The PLC generates appropriate output signals to be sent to the actuators. These actuator signals are communicated back to the simulation model via the communication interface.

Connecting every signal from the sensors in the simulation model to the inputs of the external PLC and every output of the PLC to respective actuators in the simulation model will make the process of integrating external PLC into simulation environment a very complex procedure. To overcome this problem, a controller object is developed in the simulation environment. The controller object acts as an internal PLC with inputs and outputs. Figure 3.4 describes the role of this controller object in the process of integrating external PLC into simulation environment. The simulation model consists of various objects like such as, work-pieces, sensors, actuators, conveyor systems etc. The sensors receive the necessary information from the simulation model and serves as inputs to the controller object. The controller object's function is to read the inputs and generate appropriate output signals. The outputs of the controller object are connected to the actuators. The controller object in the simulation model is not capable of processing the input signals, interpreting the instructions and generating the output signals. It merely serves as an interface to an external PLC which contains the user program necessary to control the behavior of the simulation model.

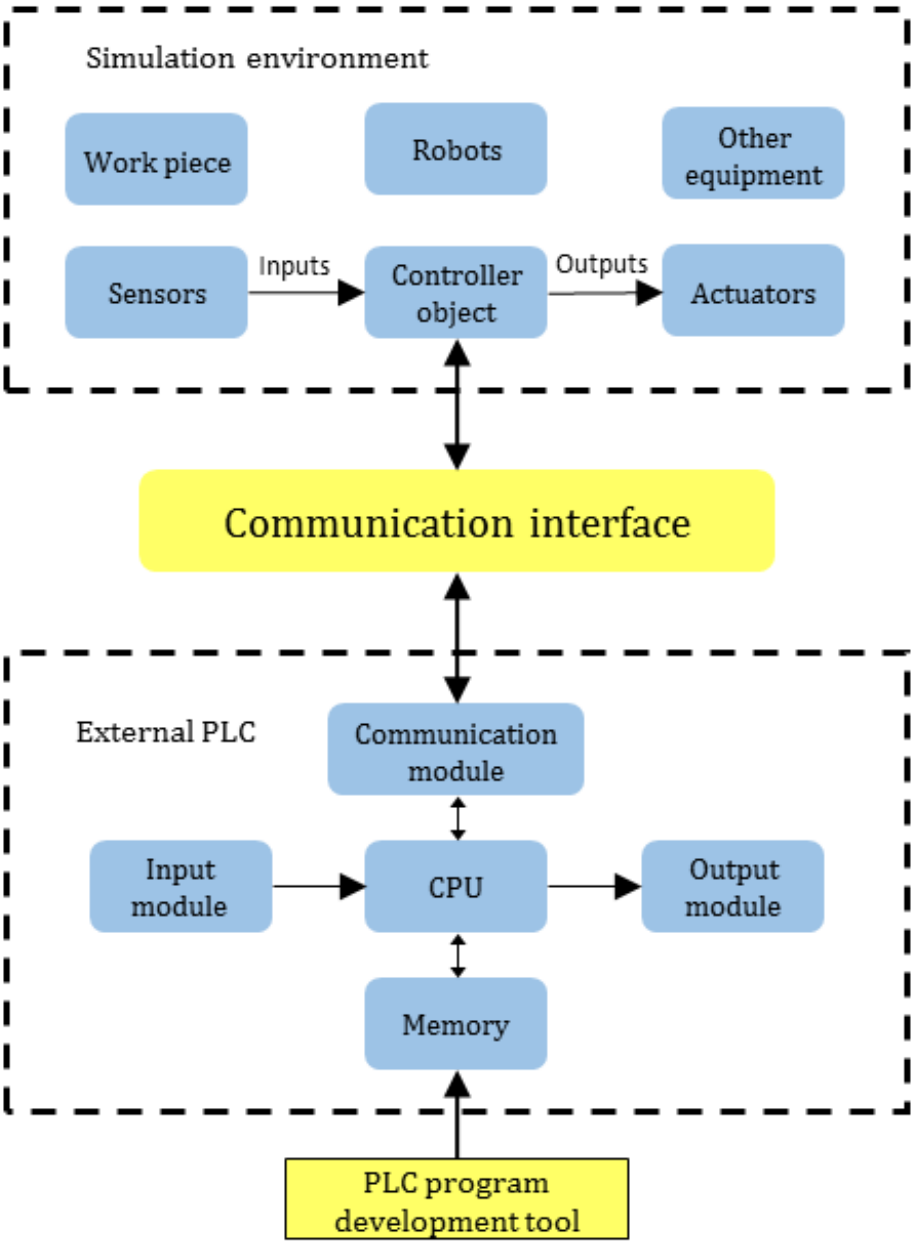


Figure 3.4: Role of controller object in the graphic simulation environment.

Communication between controller object in the simulation environment and the external PLC is established such that, the inputs of the controller object in the simulation environment are mapped to the inputs of the external PLC. The external PLC generates the output signals according to the user programs stored in its memory. The outputs are then mapped to the controller object in the simulation environment. This communication between the controller object and the external PLC can be established using various communication protocols such as COM and OPC.

The PLC programs required for controlling the mechatronic behavior of the machine in the simulated plant model are developed using a PLC program development software. Later these programs are downloaded to the memory of the external PLC.

As the main objective of this thesis is to develop a generic PLC integration concept, the controller object in the simulation environment should be able to communicate with different types of external PLCs. Figure 3.5 illustrates the concept of the generic controller object in the simulation environment. It includes inputs and outputs that connect to the sensors and actuators. The controller object can be configured to communicate with different type of external PLCs, however to avoid conflicts, only connection to one of the external PLCs be active at a given time.

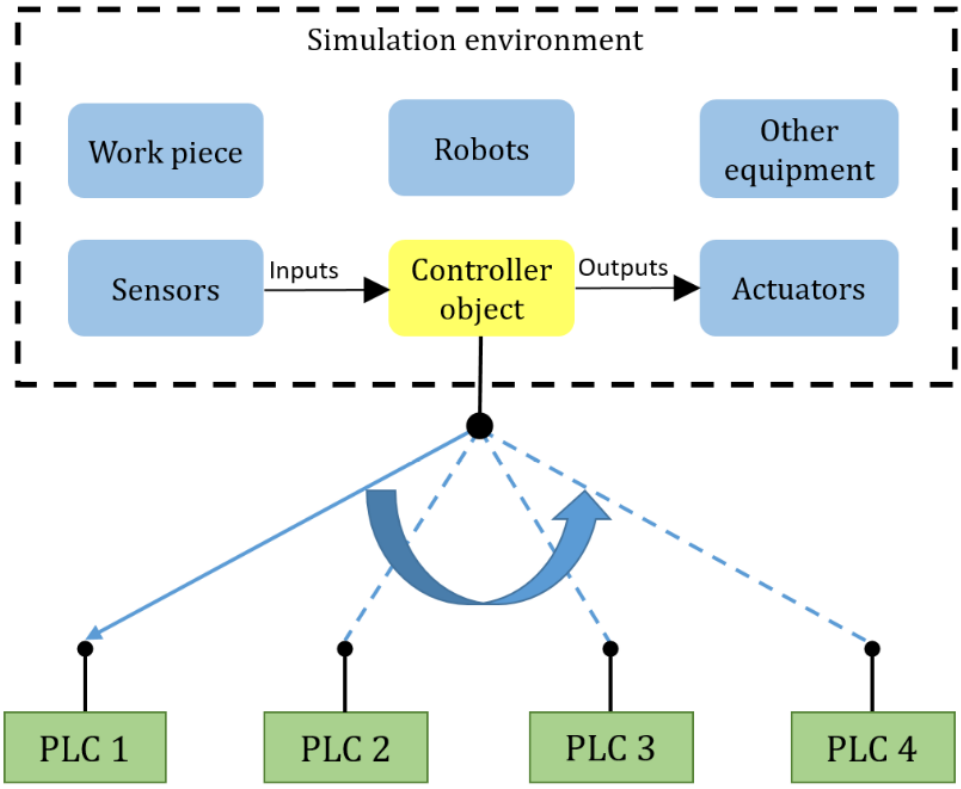


Figure 3.5: Concept of a generic controller object

The connection between the controller object in the simulation environment and external PLC is as illustrated in the figure 3.4. The inputs and outputs of the controller object are mapped to the inputs and outputs of the PLC having an active connection to the controller object.

To realize connections to more than one type of external PLC, the controller object in the simulation environment is developed as shown in the figure 3.6.

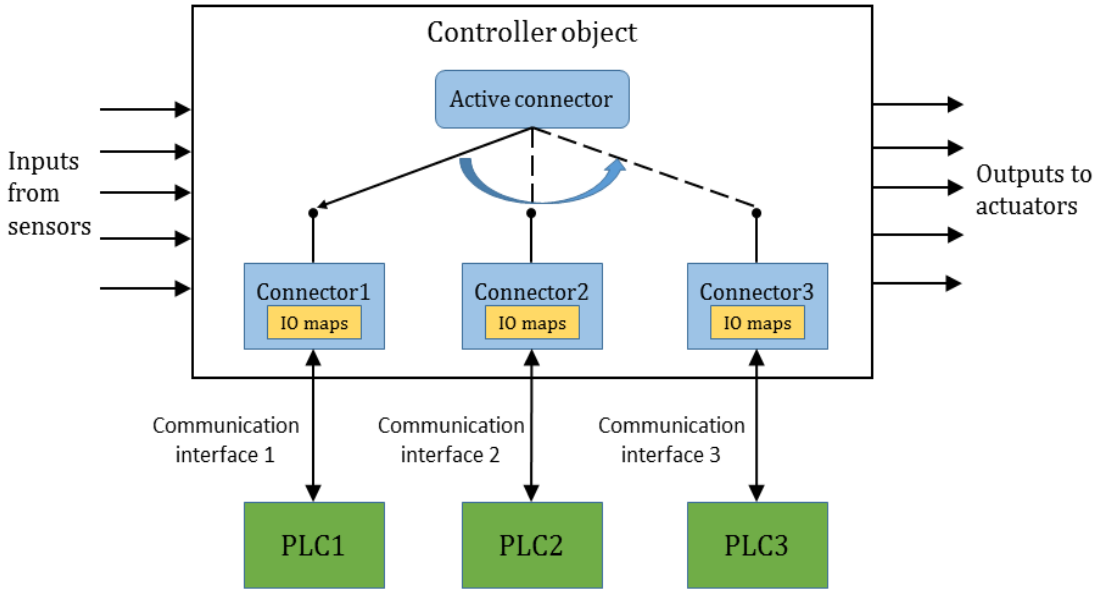


Figure 3.6: Overview of the generic controller object

To facilitate connection to more than one external PLC using a single controller object in the simulation environment, the concept of 'connectors' is introduced. Connectors are objects with inputs and outputs which communicate with the external PLCs. Connectors exchange inputs and outputs with the external PLC via a communication interface.

As sensors and actuators in the simulation model are connected to the generic controller object and the external PLC exchanges the input-output data with the connector, some kind of communication should exist between the generic controller object and connector so that the simulation model behaves in accordance with the control program running on the external PLC.

'IO maps' in the controllers as shown in the figure 3.6, perform the function of mapping inputs and outputs of the generic controller object and the connector. Each of the connector creates IO maps which map their respective inputs and outputs to the inputs and outputs of the controller object. In order to avoid the conflict, only the IO maps of the active connector are made effective during the simulation.

To demonstrate the ability of the controller object to connect to more than one external PLC, integration with two different types of PLCs is prototypically developed. This is illustrated in the figure 3.7.

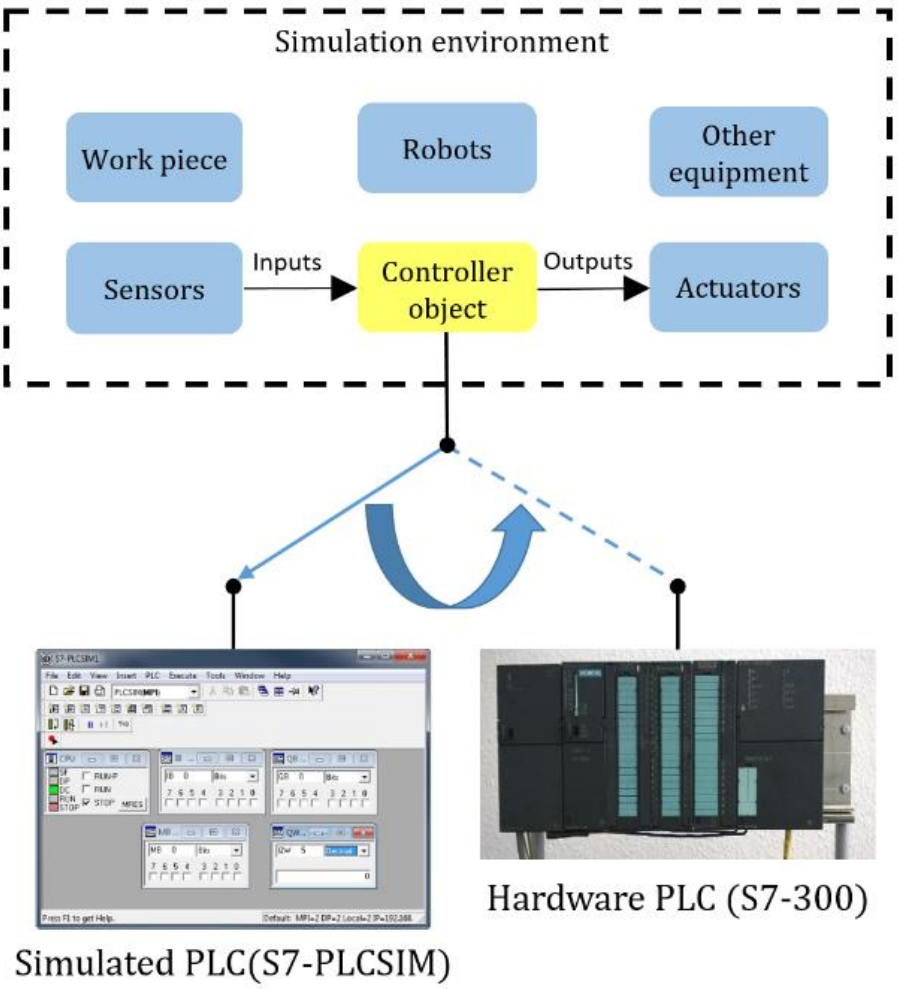


Figure 3.7: Prototype implementation of the generic controller object

Connection to external simulated PLC S7-PLCSIM and hardware PLC S7-300 is implemented in this thesis. Since the controller object is developed in a generic manner connection to any other type of PLC can be easily integrated.

4 Implementation

This chapter describes the implementation of the proposed PLC integration within a graphic simulation environment. The internal controller object as described in the previous chapter is developed in the VEROSIM environment. The controller object is capable of communicating with external PLCs. The controller object in the simulation environment is called 'PLCNode'. The PLCNode will be capable of communicating with different types of external PLCs. Currently communication interfaces with two types are implemented: simulated PLC S7-PLCSIM and hardware PLC S7-300. Figure 4.1 gives an overview of the implementation of the PLCNode.

The PLCNode has several inputs and outputs which corresponds to the inputs and outputs of the connected PLC. These inputs and outputs of the PLCNode are connected to the sensors and actuators in the VEROSIM model. Connectors configured in this PLCNode communicate with the external PLCs. As communication with two types of PLCs is implemented in this master thesis, two types of connectors can be configured in the PLCNode. From the figure 4.1, one of the connectors is of type ConnectorPLCSim, which communicates with the external simulated PLC Siemens S7-PLCSIM. The other connector is of the type ConnectorOPC, which communicates with the external hardware PLC SIMATIC S7-300 from Siemens.

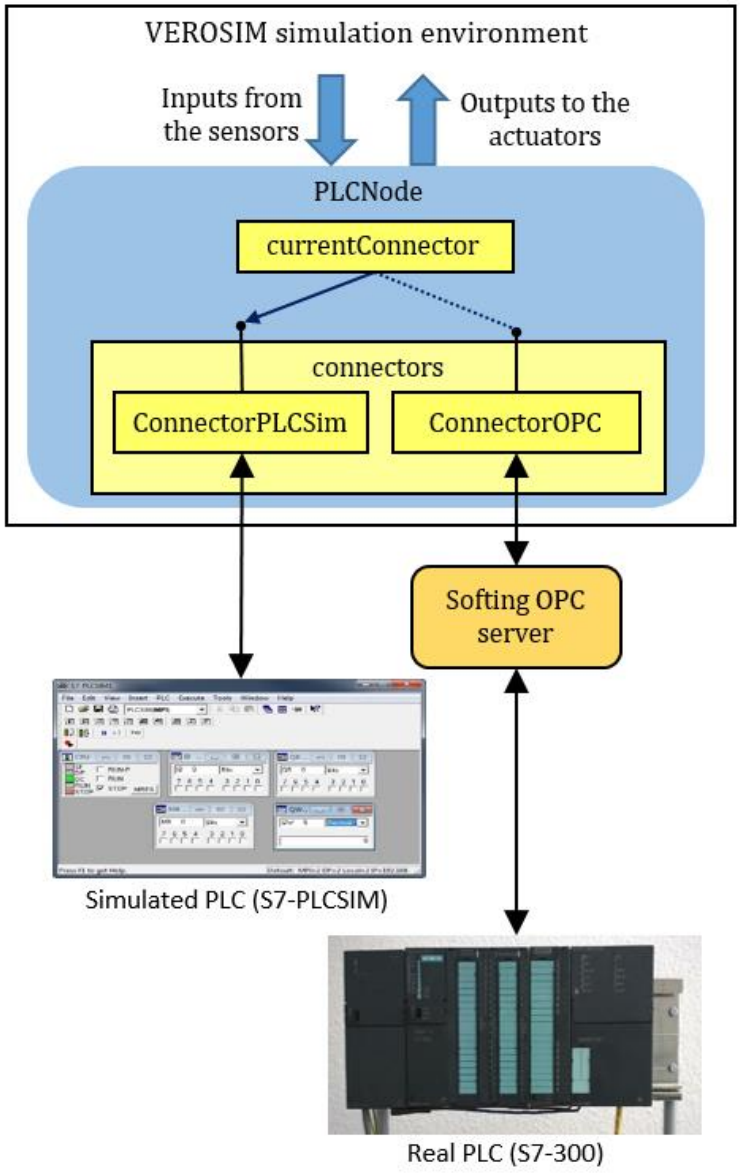


Figure 4.1: Integration of external PLCs into VEROSIM

To have a better understanding of the implementation of PLCNode it is necessary to get acquainted with some basic concepts necessary for development of new interfaces in the VEROSIM environment. Concepts in VEROSIM such as plugin, extension, reference etc. are discussed further.

VEROSIM framework is based on the concept of scene graph and the core database is organized as a directed acyclic graph. Addition of arbitrary data to this graph is based on methods in object oriented modeling. [24]

All simulation functionalities of the framework are achieved by creating specialized plugins which interact with the VSD core. Various simulation and data processing algorithms, interfaces to hardware and software systems, user interfaces etc. are implemented as plugins. Using the VSD, the plugins can communicate with the database, as well as establish directed communication between themselves. [24]

Specializations of already existing classes of 'nodes' are created through inheritance. 'Nodes' can have properties, and modeling of aspects of real world objects is easily carried out by storing arbitrary information in properties. [24]

The concept of 'extension' is introduced to cover relations other that exist between objects. "An extension can be interpreted as an active property added to a node in order to extend its capabilities". As additional degree of specialization is added to the existing object, extensions can be thought as similar to the concept of inheritance. But, in contrast to the static nature of inheritance, the extensions are dynamic allowing the specialization of the nodes to be created in runtime. Special attributes can be added to the node or removed from the node according to the situation with the help of extensions. [24]

In the VEROSIM database properties are allowed to hold 'reference' to other nodes within the database. This allows to model more complex relationships (for example, for closed loop kinematics) breaking with the acyclic nature of the basic scene graph [24].

Similar to other functionalities in VEROSIM, the PLC object is developed as a plugin interacts with the VSD core. The implementation of the PLCNode and communication to external simulated and hardware PLC is done in three distinct plugins namely *VSPluginPLCSim*, *VSPluginOPCClient* and *VSPluginPLCConnection*.

- *VSPluginPLCConnection:* The plugin VSPluginPLCConnection provides the PLCNode, which acts as an internal controller object with inputs and outputs. The creation of different connectors managed by the plugin VSPluginPLCConnection. When more than one connectors are configured in the PLCNode, the plugin enables communication with only one of the configured connectors at a given time depending on the connector in the property currentConnector.
- *VSPluginPLCSim:* The plugin VSPluginPLCSim creates a PLCSim Extension which is used by the ConnectorPLCSim. It is responsible for establishing the

communication between the PLCNode in the simulation environment and the external simulated PLC. The communication is done via the COM object S7ProSim.

- *VSPluginOPCClient:* The plugin VSPluginOPCClient creates an OPCClient node which is used by the PLCNode's ConnectorOPC. It is responsible for establishing the communication between the PLCNode in the simulation environment and the external hardware PLC. The communication with the hardware PLC is established with the help of the software OPC Server Ethernet from Softing. The data transfer between the OPC server and the external hardware PLC is done via a TCP/IP connection.

4.1 Plugin VSPluginPLCConnection

The plugin VSPluginPLCConnection is the master plugin, which manages the creation and configuration of various connectors to connect to the external PLCs. The plugin is developed from the basic idea as represented in the figure 3.6.

Figure 4.2 describes the detailed implementation of the plugin VSPluginPLCConnection. The plugin contains various classes like, PLCNode, Connector, and IOMapping. All the three classes PLCNode, Connector and IOMapping are derived from the base class VSD::Node.

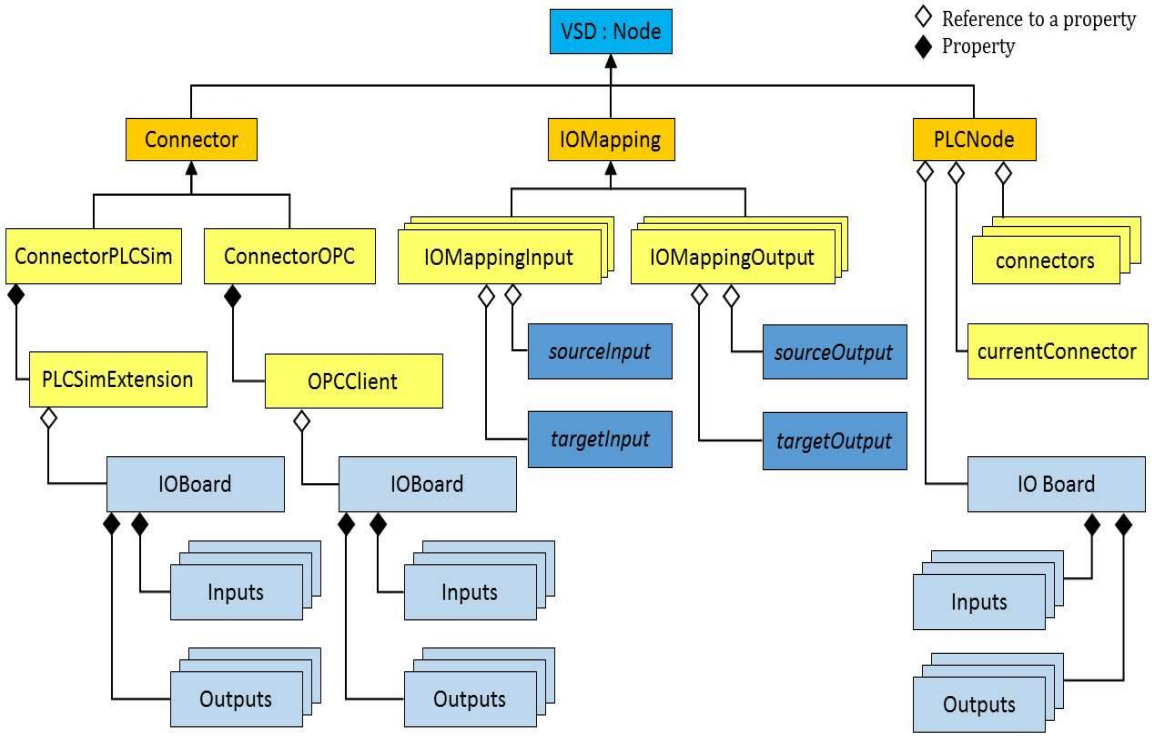
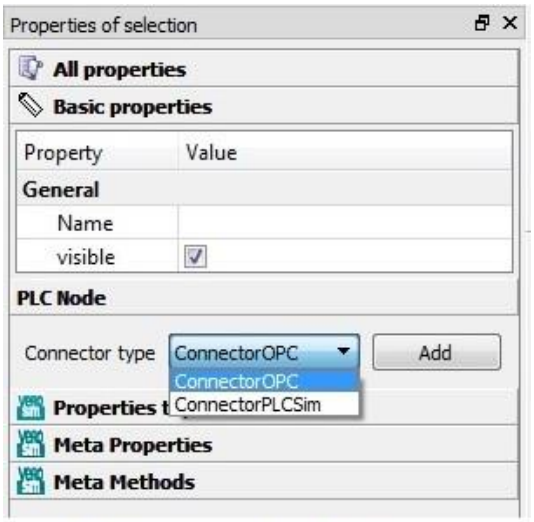
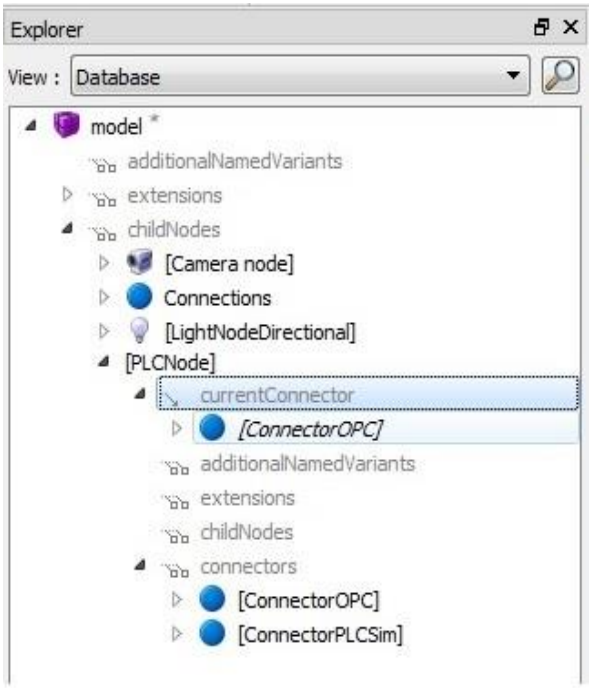


Figure 4.2: The plugin VSPluginPLCConnection

The Class PLCNode creates a PLCNode object in the VEROSIM environment. The PLCNode can be added into the VEROSIM project from the internal model library which contains a diversity of preconfigured components. The PLCNode contains an IO board, to which inputs and outputs can be added. The inputs and outputs are then connected by the user to the sensors and the actuators in the VEROSIM model. The property widget of the PLCNode contains a drop down menu with the list of all available connectors. Selecting the desired connector and clicking the button 'Add' creates a connector of that type. PLCNode has a property list 'connectors' which hold the list of connectors created by the user. Figure 4.3 (a) shows the property widget of the PLCNode and figure 4.3 (b) shows the PLCNode with two connectors added. More than one connector can be configured in the PLCNode. However, at any given time only one of the connectors can be used to connect to the external PLC. The property 'currentConnector' holds the connector which is used to connect to the external PLC at a given time. The 'currentConnector' can be changed by selecting the desired connector from the list of available connectors and dragging and dropping it on to the property 'currentConnector'.



(a) PLC Node property widget



(b) PLC Node with connectors added

Figure 4.3: Creating connectors in a PLCNode

The class Connector is the base class from which different type of connector classes are derived. Two classes are derived from the Connector class. The class ConnectorPLCSim represents a connector of type ConnectorPLCSim with an extension "ExtensionPLCSim" to connect to the external simulated PLC S7-PLCSIM. An IO board in the ConnectorPLCSim with inputs and outputs enables the exchange of data between the ConnectorPLCSim and the simulated PLC S7-PLCSim. The number and type of inputs

and outputs on the ConnectorPLCSim's IO board should be equal to that in the PLC program downloaded onto the S7-PLCSim in order to establish a successful connection between the two. In addition to the number and type of inputs and outputs, the order of the inputs and outputs organized in the PLCNode's IO board should match the order of the inputs and outputs defined in the S7-PLCSim program. In case if the inputs and outputs are not organized in the same order, the mapping of inputs and outputs will be inconsistent and the simulation model might not function as intended.

The class ConnectorOPC is also derived from the base class Connector. It represents an OPC connector with an OPCClient node which connects to the external hardware PLC via an OPC server. An IO board in the OPCClient node enables the exchange of data between ConnectorOPC and the OPC Server. The OPC Server is connected to the hardware PLC S7-300 via a TCP/IP connection. The process data from the OPC server is made available by means of OPCItem objects. All the inputs and outputs of OPCClient node include an extension OPCItem, as each process variable (input/output) on an OPC client is of type OPCItem. Each of the inputs and outputs is mapped to one of the OPCItems offered by the OPC server, via a drop down menu in the property widget. A group of OPCItems accessed by the OPC client are organized into an OPCGroup. The IO board of the OPCClient node has an extension "OPCGroup" which groups all the OPCItems accessed by the OPCClient node.

The PLCNode and the connectors, ConnectorPLCSim and ConnectorOPC, each have distinct IO boards. The PLCNode IO board connects to the sensors and actuators in the VEROSIM model while the IO boards in the Connectors ConnectorPLCSim and ConnectorOPC, enables the exchange of data between the connectors and the external simulated and hardware PLC respectively. For the simulation model to work with the external PLCs a mapping of inputs and outputs of the PLCNode IO board to the inputs and outputs of the IO board of the connectors is required.

The Class IOMapping is responsible for the mapping of the inputs and outputs of the PLCNode to the inputs and outputs of the configured connectors. Two separate classes IOMappingInput and IOMappingOutput are derived from the class IOMapping to map the inputs and outputs respectively. The class IOMapping is derived from the VEROSIM base class VSS::TaskStep that executes the IOMapping function every simulation cycle.

Thus the mapping of inputs and outputs of the PLCNode and the connector is done every simulation cycle.

The class IOMappingInput, derived from the base class IOMapping, maps the sourceInput to the targetInput. As seen from figure 4.4, the sourceInput is an input from the PLCNode's IO board and the targetInput is an input from the connector's IO board. The class IOMappingOutput, also derived from the base class IOMapping, maps the sourceOutput to the targetOutput. The sourceOutput is an output from the connector's IO board and the targetOutput is an output from the PLCNode's IO board.

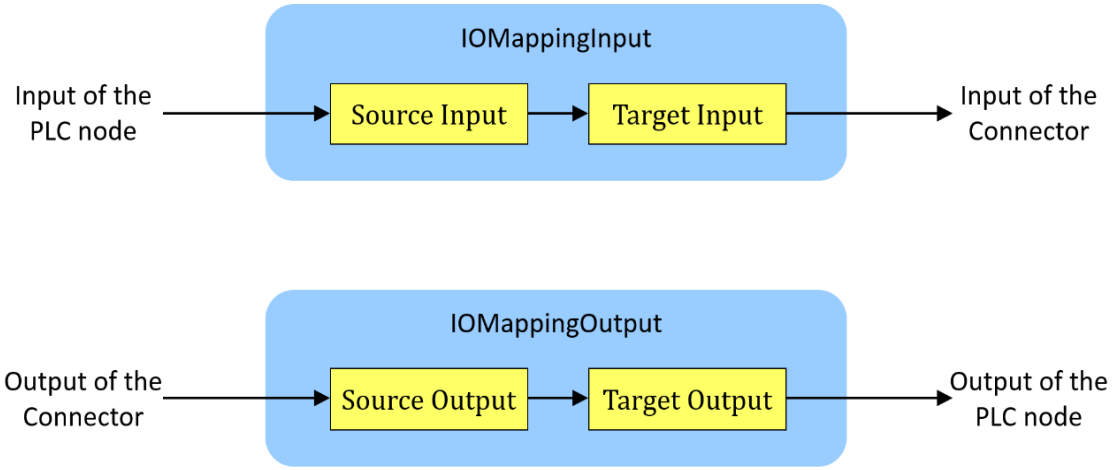


Figure 4.4: Implementation of the class IOMapping

IO maps created for each of the inputs and outputs are stored as a list in the property 'nodesIOMapping' of the connector object. Thus each connector configured will have a list of IO mappings in the property 'nodesIOMapping'. After adding the inputs and outputs to the IO boards of PLCNode and the connector, IO maps can be created by clicking on the button 'Create IO mappings' as shown in the figure 4.5. Since more than one connector can be configured by the user, not all the connector's IO maps should be active at the same time. To solve this problem, each of the connectors checks for property 'currentConnector' of its parent PLCNode. The IO maps of the connector are made active only if the connector is selected to be the 'currentConnector', else the IO maps of the connector are inactive.

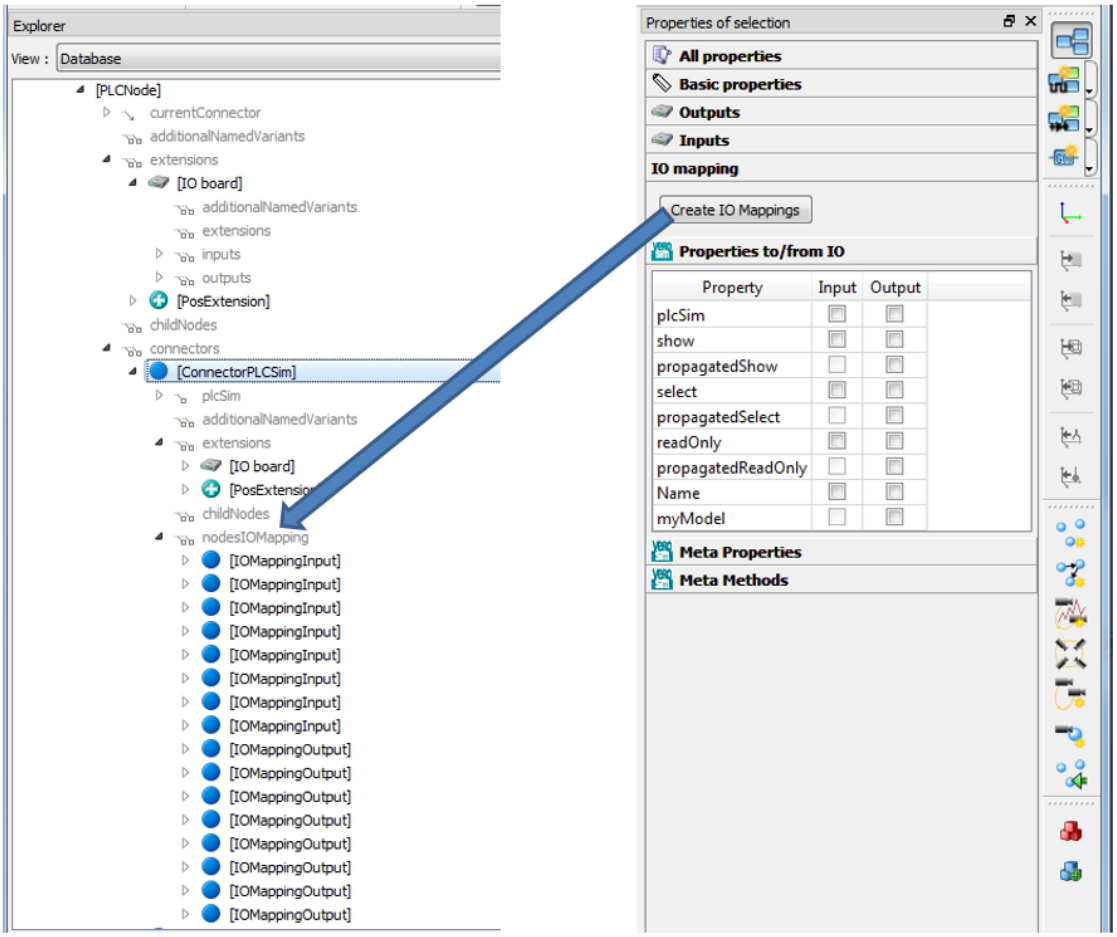


Figure 4.5: Creating IO maps

In case of the ConnectorPLCSim, the exchange of data between ConnectorPLCSim and the simulated PLC S7-300 is dependent on the order of the inputs and outputs. This dependency is used when automatically creating the IO maps between the ConnectorPLCSim and PLCNode. Provided the number of inputs and outputs in the two IO boards of the ConnectorPLCSim and PLCNode are equal, clicking the 'Create IO Mappings' button on the property widget of the ConnectorPLCSim automatically creates the IO mappings for the inputs and the outputs.

As each of the inputs and outputs in the ConnectorOPC's IO board are mapped to the OPCItem provided by the OPC server by selecting the OPCItem, the IO mapping between the ConnectorOPC and PLCNode is not done automatically. However for the user's convenience, IO maps (IOMappingInput and IOMappingOutput) are created depending on the number of inputs and outputs in the PLCNode's IO board on clicking the button 'Create IO Mappings'. In case of the IO maps for inputs, the IOMappingInput

objects are created with the sourceInputs automatically filled. The IO maps for outputs, IOMappingOutput are created with the targetOutputs automatically filled. Adding the targetInputs and the sourceOutputs has to be done manually by the user.

4.2 Plugin VSPluginPLCSim

The plugin VSPluginPLCSim creates the connection to the external simulated PLC S7-PLCSIM via the COM object S7ProSim. By adding a reference to the S7ProSim COM object, the methods provided by this object can be used to get programmatic access to the S7-PLCSIM. The communication sequence between the VSPluginPLCSim and S7-PLCSIM via the S7ProSim COM object is shown in figure 4.6.

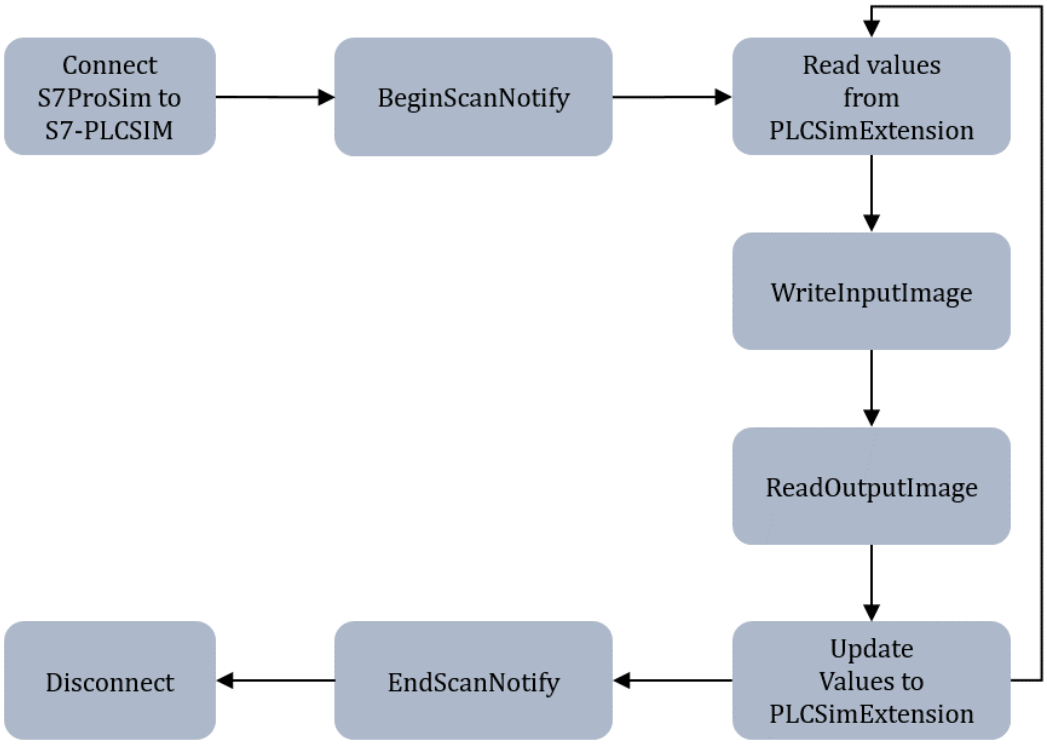


Figure 4.6: Communication with S7-PLCSIM

- The method ‘Connect’ connects S7ProSim COM object to the first instance of S7-PLCSIM which has the instance number 1.
- The method ‘BeginScanNotify’ registers S7ProSim for callbacks from the controller.
- The input values from the connector object in the VEROSIM environment are copied to a local variable.

- The method ‘WriteInputImage’ writes PLCSim connector’s input values to the peripheral inputs of the S7-PLCSIM.
- The method ‘ReadOutputImage’ is used to read the elements from the peripheral outputs of S7-PLCSIM.
- These output values are written back to the outputs of the PLCSim connector in the VEROSIM environment.
- The method ‘EndScanNotify’ unregisters S7Prosim for the call backs from the controller. The events ScanFinished and PLCSimStateChanged are no longer sent.

4.3 Plugin VSPluginOPCClient

The plugin VSPluginOPCClient is able to create an OPCClient object and connect to the OPC server. The OPC server fetches the data from the external hardware PLC S7-300 via a TCP/IP connection. Figure 4.7 shows the configuration and working of the plugin VSPluginOPCClient.

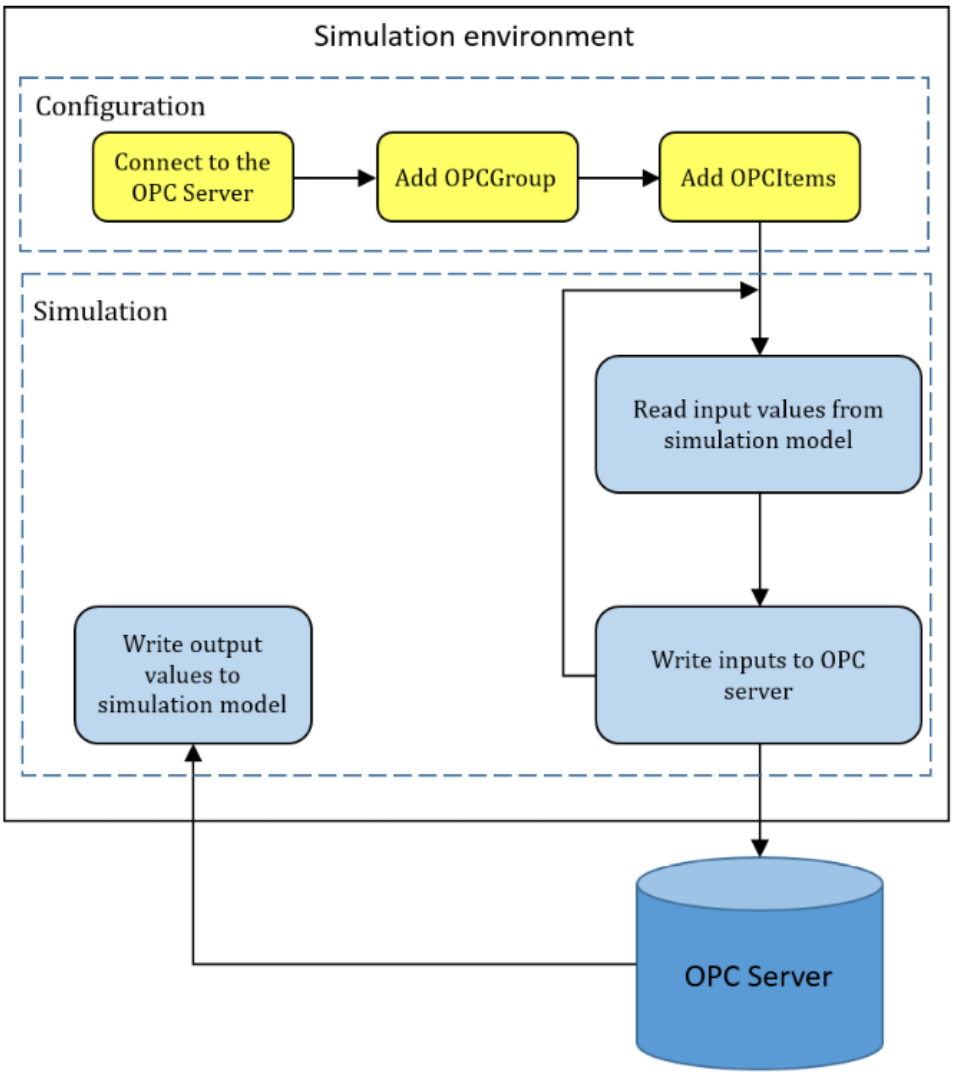


Figure 4.7: Communication with S7-300 via OPC client-server

The communication of the OPC Client object in the ConnectorOPC with the external hardware PLC S7-300 is done in two steps: configuration and simulation.

In the configuration step, the OPCClient object is connected with the available OPC server i.e. *INAT TCPIPH1 OPC* Server from Softing. After connecting to the OPC server, an OPCGroup is created in the OPCClient node. The OPCGroup holds the list of variables accessed from the OPC server. OPCItems accessed by the OCClient are then added into the OPCGroup.

The simulation step includes the process of data exchange between the OPC client and server. The input values at the OPCClient nodes IO board are read and if there is any

change in the input variables they are written to the OPC server. This process is repeated cyclically. The communication of data from the OPC server to the VEROSIM model is asynchronous in nature. The outputs from the OPC server are asynchronously read by the outputs of the VEROSIM OPCClient node.

5 Validation

The operation of the controller object created in the VEROSIM environment is validated with the help of a mechatronic model. This chapter describes the development of the simulation model in VEROSIM and the PLC program which controls the behavior of the model. Further, the process of creating the PLC object in the VEROSIM environment, and configuring different connectors to connect to the external simulated and hardware PLC is illustrated.

5.1 Simulation Model

Figure 5.1 shows the simulation model developed in VEROSIM. This is a mechatronic model where the work-piece is placed on a work-piece carrier and moved between two stations, station 1 and station 2 by a conveyor belt arrangement. At station 1 a robot arm grips the work-piece and rotates it by 180 degrees. At station 2 the work-piece waits for 3 seconds.

The CAD data for the simulation model is imported into the VEROSIM environment and mechanisms for the conveyor belts, work-piece and the robot arm are added. The 'Start/Stop' button turns on the conveyor belts. Two light barrier sensors are used to detect the arrival of the work-piece carrier at the two stations. Four positions for the robot arm are defined in order to perform the rotation action. The four robot arm positions include two translation positions namely T_Pose0 and T_Pose1 and, two rotational positions namely R_Pose0 and R_Pose1. A 'simple robot controller' from the VEROSIM model library moves the robot arm from one position to the other. The movement sequence of the robot arm is controlled by the PLC program. Each position of the robot arm has an input, when triggered the simple robot controller moves the robot arm to that position. As the robot arm reaches the desired position, an output signal 'reached' is generated. Thus for all the four robot positions the input signals are set by the PLC and the output signals are sent to the PLC as an acknowledgement. The waiting time for the work-piece at station 2 is monitored by a timer in the PLC program.

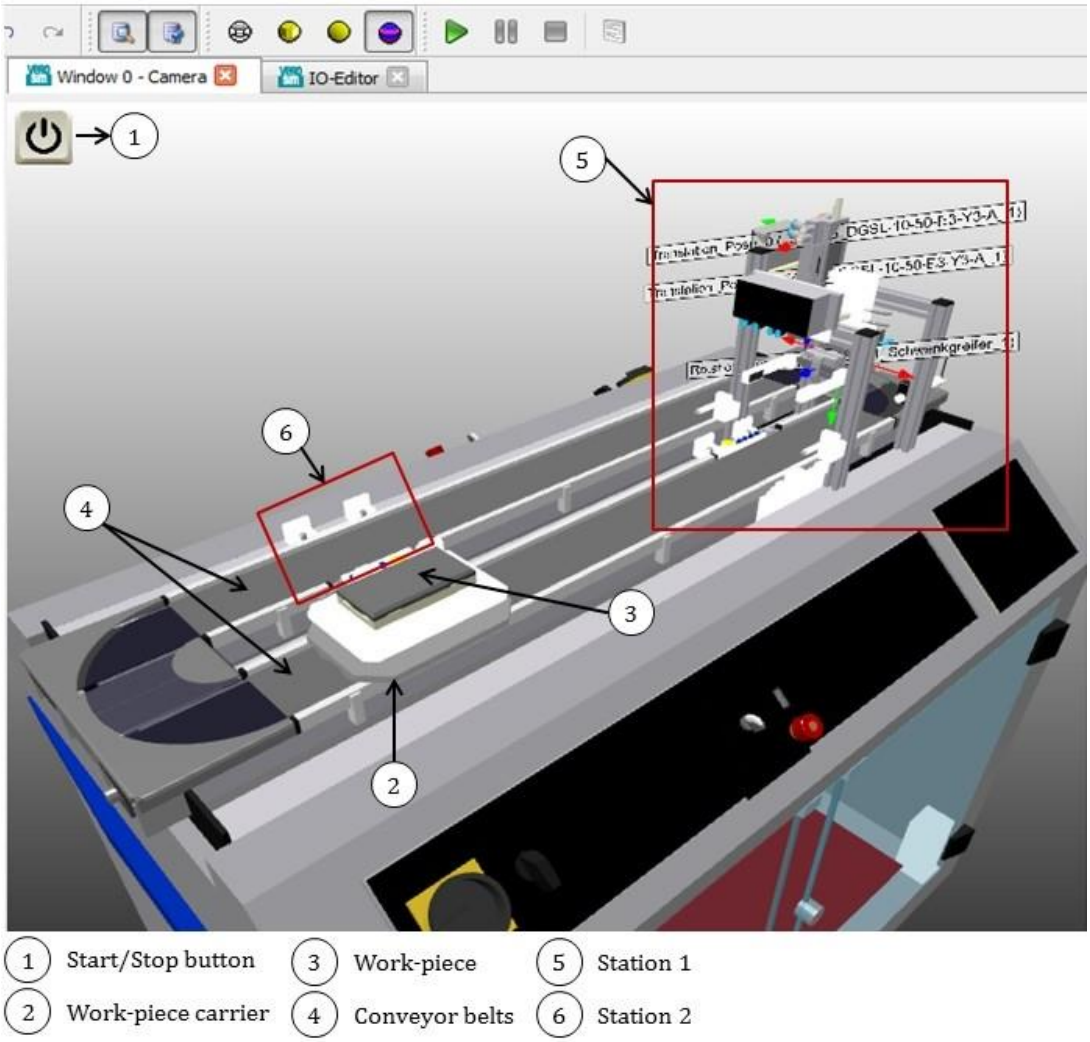


Figure 5.1: VEROSIM model

Working of the 3D model:

1 Pressing the ‘Start/Stop’ button on the model, starts the conveyor belts.

2 When the object reaches the station 1 the conveyor belts are stopped and the robot at station 1 grips the work-piece and rotates it by 180 degrees and places it back on the work-piece carrier.

3 The conveyor belts are then started.

4 When the work-piece carrier reaches the station 2, the conveyor belts are stopped for 3 seconds and started again.

5 Back to step 2 or pressing the ‘Start/Stop’ button stops the operation of the model.

The operation of the model is controlled by PLC programs. The inputs from the model are connected to the inputs of the PLC. The PLC generates appropriate output signals depending on the user program stored in the PLC memory. The outputs of the PLC are connected to the actuators in the VEROSIM model. Thus, before writing the PLC programs for the model, a list of inputs and outputs used by the PLC has to be defined.

Output	**Description**
Start_Conveyor	Start the conveyor belts
T_Pose0_input	Input signal for T_Pose0
T_Pose1_input	Input signal for T_Pose1
R_Pose0_input	Input signal for R_Pose0
R_Pose1_input	Input signal for R_Pose1
Grip	Grip the work-piece
Release_Gripper	Release the gripper

Table 5.1: List of inputs from the VEROSIM model

Input	**Description**
Start	Signal generated on pressing the Start/Stop button
Station1_reached	Work-piece carrier reached station 1
Station2_reached	Work-piece carrier reached station 2
T_Pose0_reached	Robot arm reached the position T_Pose0
T_Pose1_reached	Robot arm reached the position T_Pose1
R_Pose0_reached	Robot arm reached the position R_Pose0
R_Pose1_reached	Robot arm reached the position R_Pose1
ObjectGripped	Object gripped by the robot

Table 5.2: List of outputs from the VEROSIM model

5.2 PLC Programming

The SIMATIC Step 7 software from Siemens is used for writing the PLC programs. A sequential control system is developed using the S7-Graph package of the STEP 7 software. The list of inputs and outputs created for the controller object in the simulation model serves as inputs and outputs for the external PLC as well.

The working of the simulation model is described by a S7-Graph in the form of a sequencer. A sequencer represents a sequence of single steps and conditions that control how the process moves on to the next step. Before creating the program for the sequencer, the structure of the sequencer is specified by breaking down the working of the model into single steps as shown in figure 5.2. The structure of the sequencer is specified by the following the steps:

1 The working of the simulation model is broken down into steps and the order of the steps is specified.
2 For each step, the actions that must be performed in that step are specified.
3 Then for every step, the conditions are decided that must be met so that the process can move on to the next step.

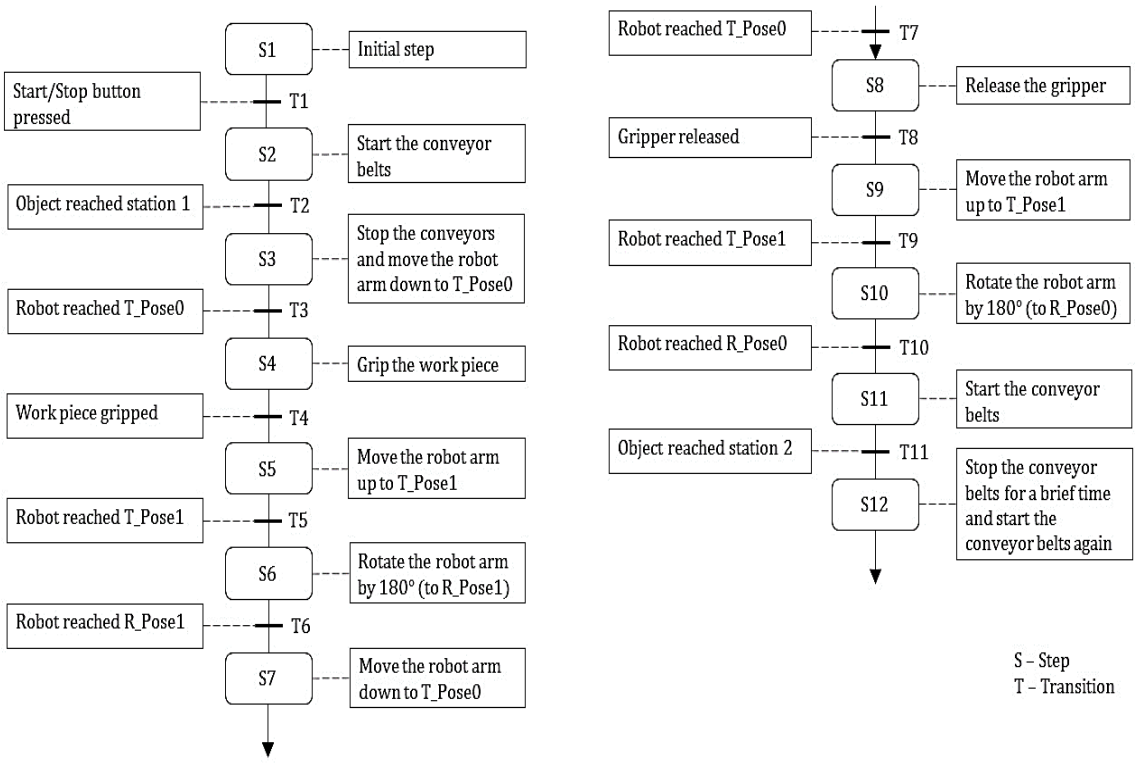


Figure 5.2: Structure of sequencer for the PLC program to control the model

Once the structure of the sequencer is developed, a new project is created in SIMATIC manager. In the new project hardware configuration for the PLC is done to be able to download the programs developed to the PLCs.

Since a rack type PLC is used, the hardware configuration begins with adding a rack. In the first slot a power supply module is added. A CPU is added in the second slot, following the CPU input and output modules are added. Lastly, a communication module is added for downloading the programs to the PLCs.

For the simulated PLC, hardware configuration is not a critical part of the project. The configuration of the project for simulated PLC includes a standard rack, CPU315-2 PN/DP(1) and a digital input/output module (DI8/DO8xDC24V/0.5A). No communication model is added in the hardware configuration for PLCSIM. Downloading the programs to the simulated PLC S7-PLCSIM is via MPI connection. Figure 5.3 shows the hardware configuration for using S7-PLCSIM.

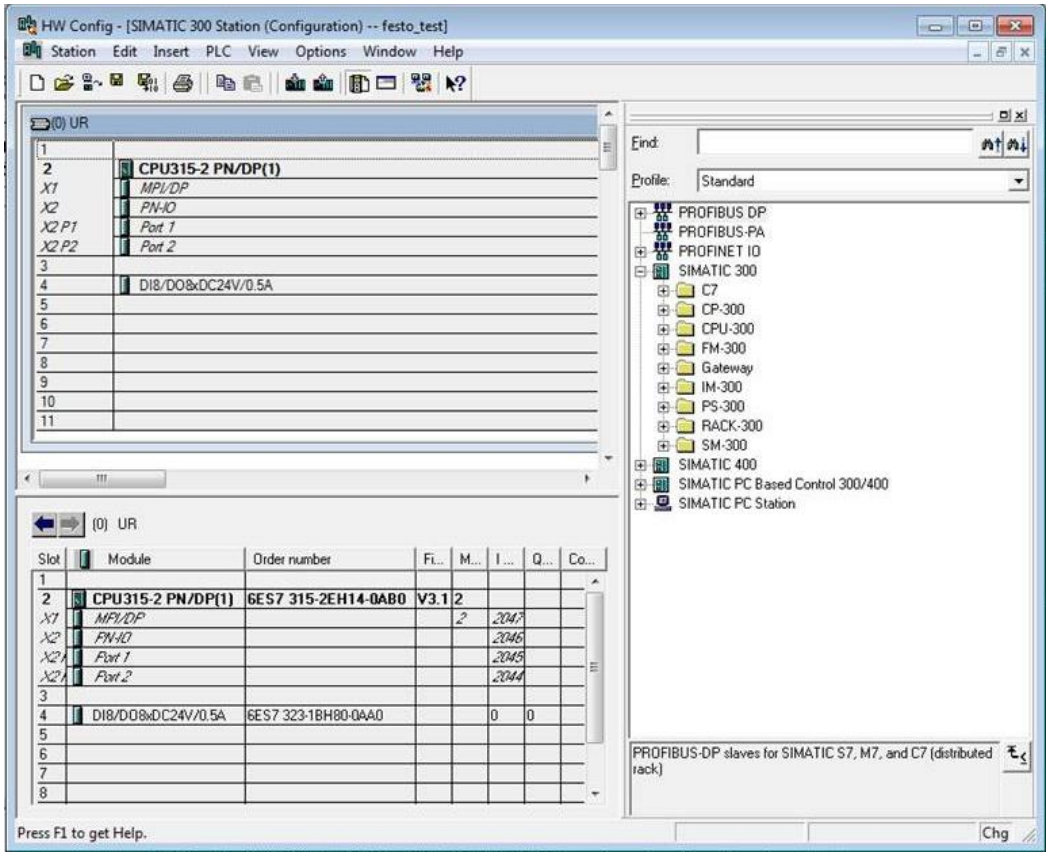


Figure 5.3: Hardware configuration for S7-PLCSIM

In case of the hardware PLC, the hardware configuration is a crucial part of the project. The types of hardware components added in the hardware configuration should exactly be same as the actual hardware. The hardware configuration of the project for the real PLC is as follows: standard rack, power supply (order number), CPU – 314-2 PN/DP (), input/output module (order number) and a communication module cp-343 advanced. Figure 5.4 shows the hardware configuration for the PLC S7-300.

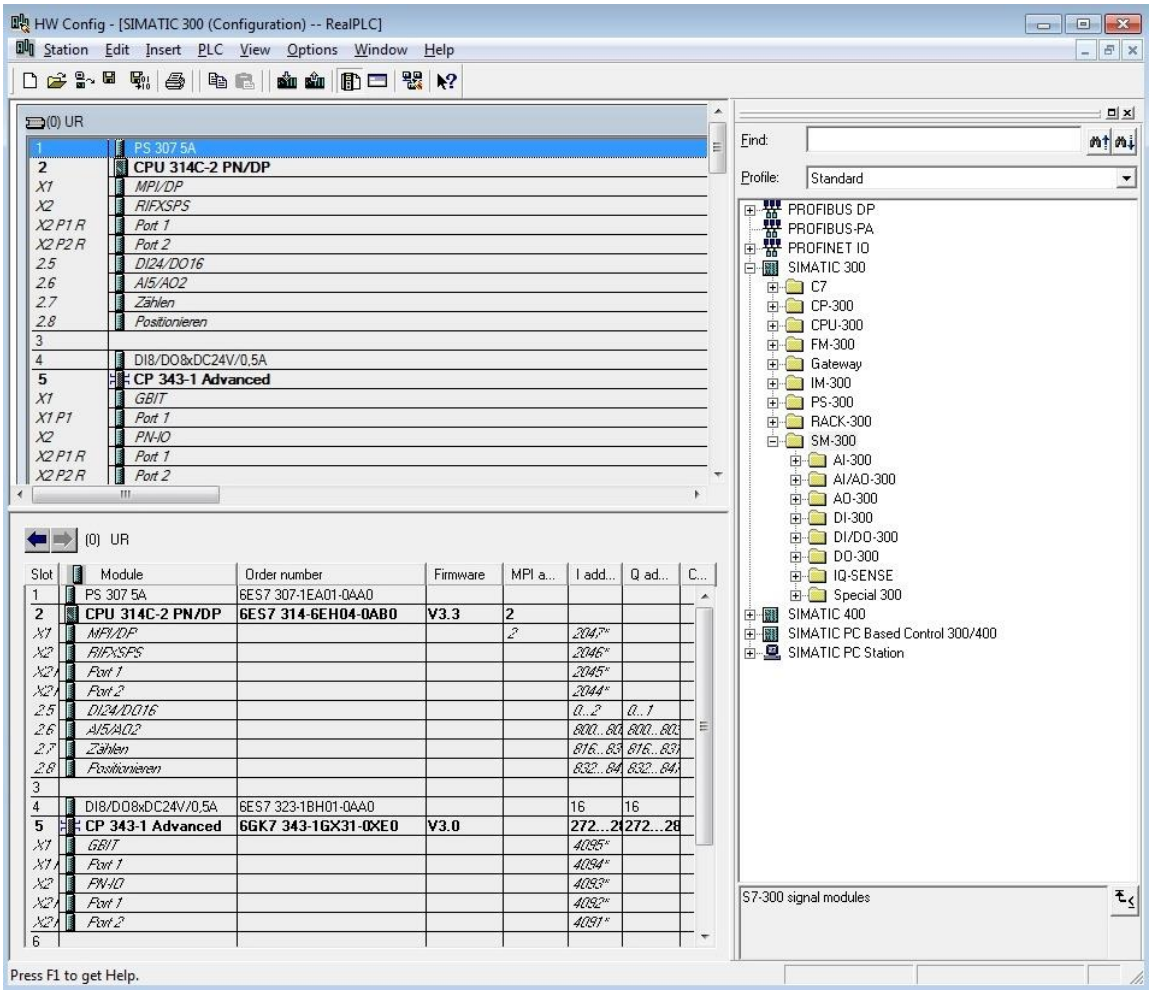
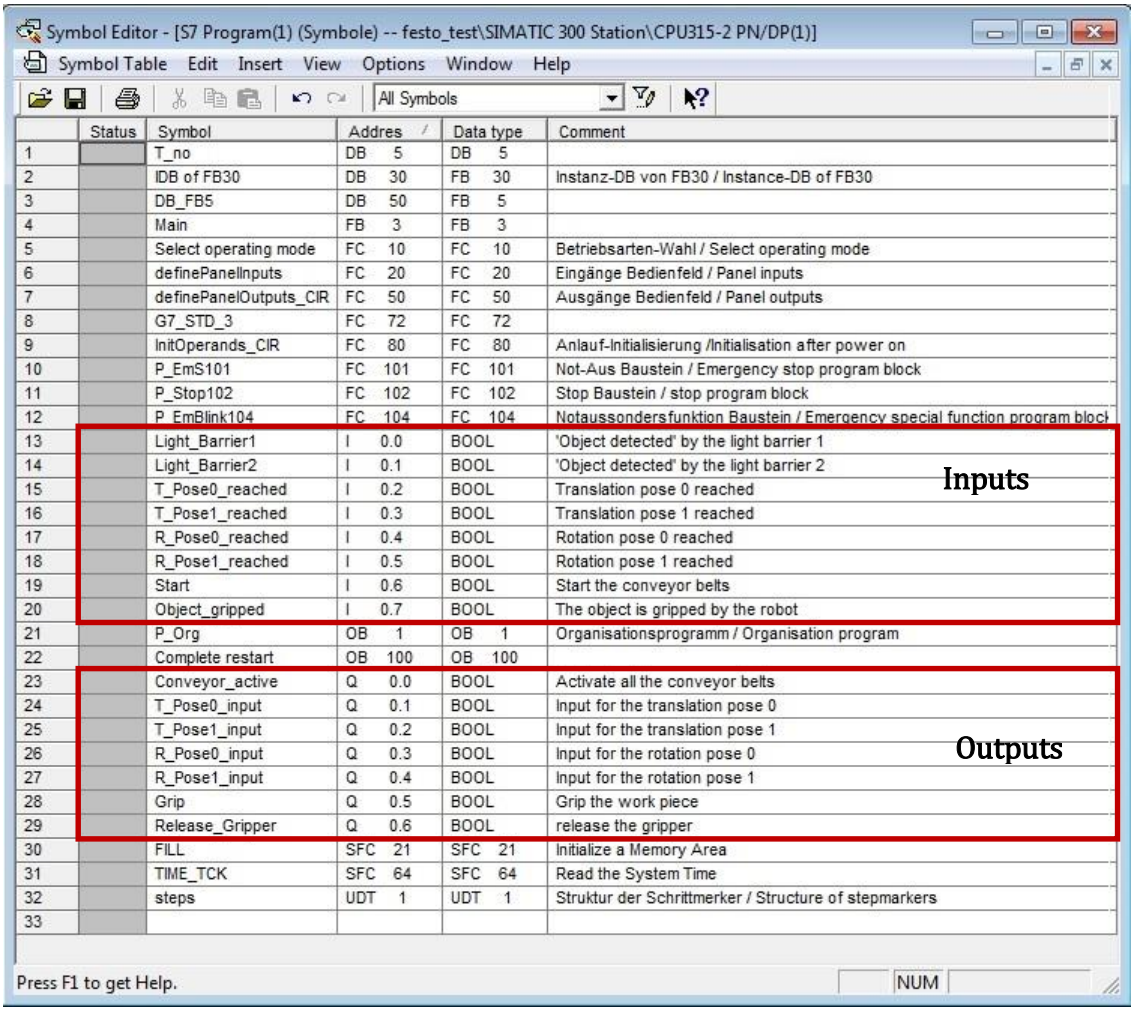


Figure 5.4: Hardware configuration for PLC S7-300

While programming in STEP 7, the I/O signals, memory bits, counters, timers, data blocks and function blocks can all be accessed by means of absolute addressing. However it is easier to read and write the program if symbols are used instead of absolute addressing. Using the 'Symbol table' name, absolute address, data type and comment can be added for every address used. All the inputs and outputs listed earlier are defined in the symbol table as shown in figure 5.5.



	Status	Symbol	Addres	Data type	Comment
1		T_no	DB 5	DB 5	
2		IDB of FB30	DB 30	FB 30	Instanz-DB von FB30 / Instance-DB of FB30
3		DB_FB5	DB 50	FB 5	
4		Main	FB 3	FB 3	
5		Select operating mode	FC 10	FC 10	Betriebsarten-Wahl / Select operating mode
6		definePanelInputs	FC 20	FC 20	Eingänge Bedienfeld / Panel inputs
7		definePanelOutputs_CIR	FC 50	FC 50	Ausgänge Bedienfeld / Panel outputs
8		G7_STD_3	FC 72	FC 72	
9		InitOperands_CIR	FC 80	FC 80	Anlauf-Initialisierung /Initialisation after power on
10		P_EmS101	FC 101	FC 101	Not-Aus Baustein / Emergency stop program block
11		P_Stop102	FC 102	FC 102	Stop Baustein / stop program block
12		P_EmBlink104	FC 104	FC 104	Notaussondersfunktion Baustein / Emergency special function program block
13		Light_Barrier1	I 0.0	BOOL	'Object detected' by the light barrier 1
14		Light_Barrier2	I 0.1	BOOL	'Object detected' by the light barrier 2
15		T_Pose0_reached	I 0.2	BOOL	Translation pose 0 reached
16		T_Pose1_reached	I 0.3	BOOL	Translation pose 1 reached
17		R_Pose0_reached	I 0.4	BOOL	Rotation pose 0 reached
18		R_Pose1_reached	I 0.5	BOOL	Rotation pose 1 reached
19		Start	I 0.6	BOOL	Start the conveyor belts
20		Object_gripped	I 0.7	BOOL	The object is gripped by the robot
21		P_Org	OB 1	OB 1	Organisationsprogramm / Organisation program
22		Complete restart	OB 100	OB 100	
23		Conveyor_active	Q 0.0	BOOL	Activate all the conveyor belts
24		T_Pose0_input	Q 0.1	BOOL	Input for the translation pose 0
25		T_Pose1_input	Q 0.2	BOOL	Input for the translation pose 1
26		R_Pose0_input	Q 0.3	BOOL	Input for the rotation pose 0
27		R_Pose1_input	Q 0.4	BOOL	Input for the rotation pose 1
28		Grip	Q 0.5	BOOL	Grip the work piece
29		Release_Gripper	Q 0.6	BOOL	release the gripper
30		FILL	SFC 21	SFC 21	Initialize a Memory Area
31		TIME_TCK	SFC 64	SFC 64	Read the System Time
32		steps	UDT 1	UDT 1	Struktur der Schrittmerker / Structure of stepmarkers
33					

Figure 5.5: Symbol table

A S7-Graph function block (FB) is created in ‘Blocks’ folder in the SIMATIC manager. In this FB the program for the sequencer is entered according to the sequencer design as shown in figure 5.2. The sequential control program for the simulation model is called and started in organization block ‘OB1’. The OB1 is created in the ladder logic (LAD) language. All the blocks in the ‘Blocks’ folder are downloaded to the PLC. Figure 5.6 shows the project created in SIMATIC Manager. The sequencer is implemented in the function block FB5 and it is called from the organization block OB1.

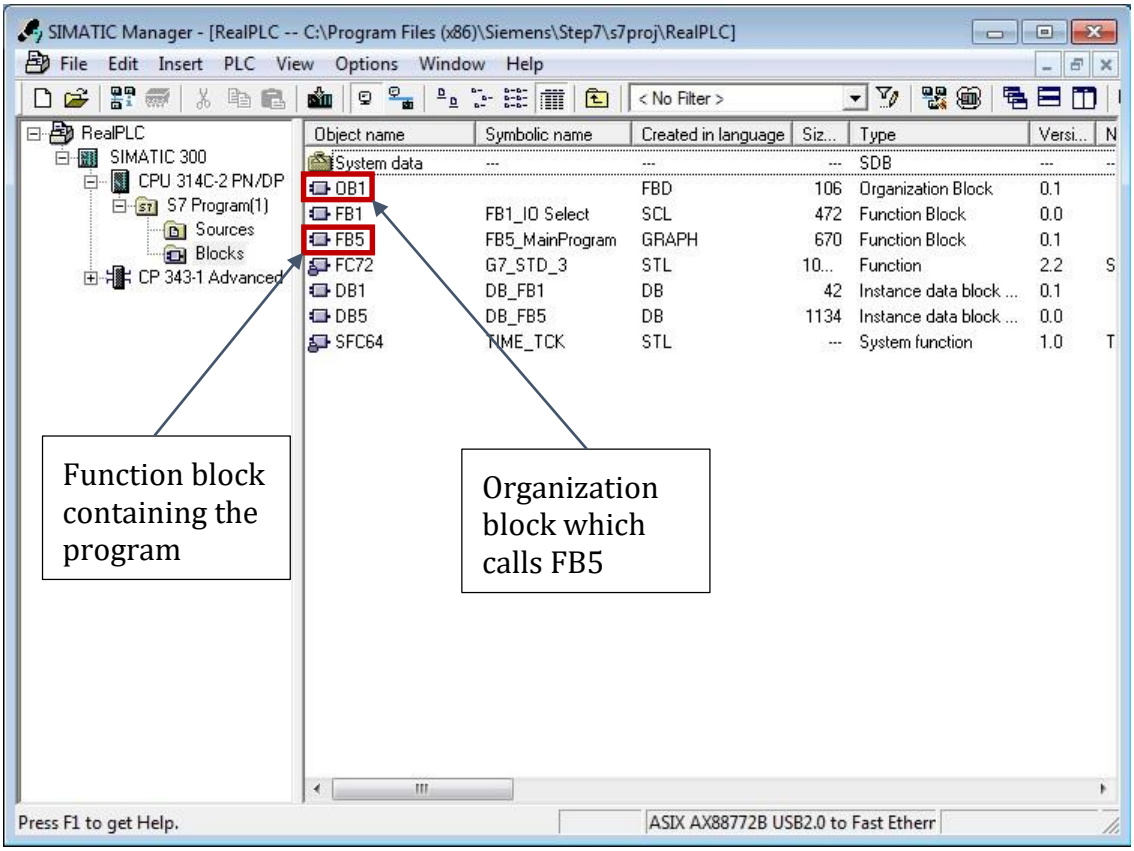


Figure 5.6: Project in SIMATIC Manager

5.3 Creating the Controller Object in VEROSIM Project

As the simulation model and the PLC programs are both implemented, the next task is to integrate the external PLC containing the control program with the simulation model. The controller object developed in VEROSIM helps in establishing the communication with the external PLC. The controller object named 'PLCNode' is added to the VEROSIM project from the VEROSIM model library. As 8 input and output signals are defined for the simulation model, 8 digital inputs and outputs are added to the IO board of PLCNode.

The VEROSIM IO editor is used to make the connections to the PLCNode from the sensors and actuators in the simulation model. Referring to the lists of inputs and outputs created in chapter 5.1, inputs and outputs from the simulation model are connected to the corresponding inputs and outputs of the PLCNode.

Connectors to connect to different types of external PLC are created in the PLCNode by selecting the type of connector and clicking the ‘Add’ button from the property widget of the PLCNode. A ConnectorPLCSim is created to connect to the S7-PLCSIM and a ConnectorOPC is created to connect to the PLC S7-300.

Configuring the ConnectorPLCSim:

The ConnectorPLCSim is created with an ‘ExtensionPLCSim’ which facilitates the communication with the external simulated PLC S7-PLCSIM. The ConnectorPLCSim requires an IO board with inputs and outputs for the communication. The ConnectorPLCSim is configured in the following manner:

- An IO board is added in the extensions of the ConnectorPLCSim.
- Similar to the PLCNode’s IO board 8 digital inputs and outputs are added to the ConnectorPLCSim’s IO board.
- Clicking the ‘Create IO Mappings’ button on the ConnectorPLCSim’s property widget creates a list of IOMappingInput and IOMappingOutput for inputs and outputs in the property nodesIOMapping. As the source and the target inputs and outputs are added automatically, the index of the inputs and outputs at the PLCNode IO board should be similar to the index of the inputs and outputs at the ConnectorPLCSim IO board.

Figure 5.7 shows the configured ConnectorPLCSim. The IO board of PLCNode consists of 8 digital inputs and 8 digital outputs, thus 8 IOMappingInput and 8 IOMappingOutput are created in the property ‘nodesIOMapping’. The sourceInput refers to the input on the PLCNode IO board and the targetInput refers to the input on the ConnectorPLCSim IO board.

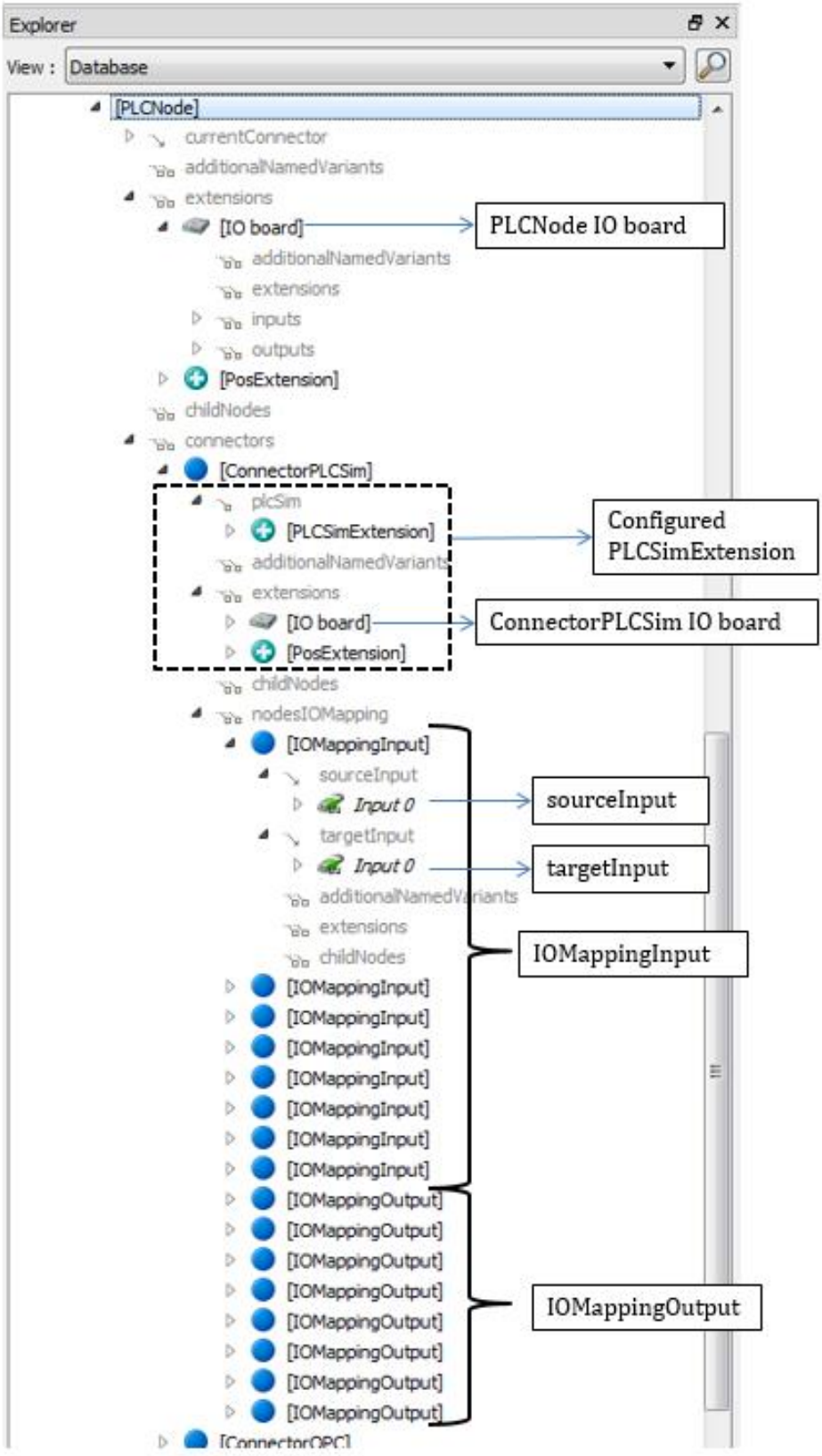


Figure 5.7: Configured ConnectorPLCSim

Configuring the ConnectorOPC:

An OPCClient object is created in the ConnectorOPC For communication with the external hardware PLC S7-300, the ConnectorOPC requires an IO board with inputs and outputs for the communication. The ConnectorOPC is configured in the following manner:

- An IO board with an extension OPCGroup is added in the OPCClient object of the ConnectorOPC.
- The OPCItems of interest are added into the OPCGroup as inputs and outputs. The OPCItems includes the inputs and outputs from the IO list created in chapter 5.1 and any other additional process variable if required.
- Clicking the 'Map IOs' button from the ConnectorOPC's property widget creates the IO maps. The number of IOMappingInput and IOMappingOutput created corresponds to the number of inputs and outputs in the PLCNode IO board. In case of ConnectorOPC the created IO maps are not complete. The inputs of the PLCNode IO board are added as the sourceInput in the IOMappingInput, and the outputs of the PLCNode IO board are added as the targetOutput in the IOMappingOutput automatically.
- The targetInputs in the IOMappingInput are added to map the inputs of the PLCNode to the inputs of the OPCClient IO board.
- The sourceOutput in the IOMappingOutput are added to map the outputs of the OPCClient to the outputs of the PLCNode IO board.

Figure 5.8 shows the configured ConnectorOPC. The IO board of PLCNode consists of 8 digital inputs and 8 digital outputs, thus 8 IOMappingInput and 8 IOMappingOutput are created in the property 'nodesIOMapping'. The sourceInput refers to the input on the PLCNode IO board and the targetInput refers to the input on the ConnectorPLCSim IO board. The sourceInput is added automatically and the targerInput is configured by the user.

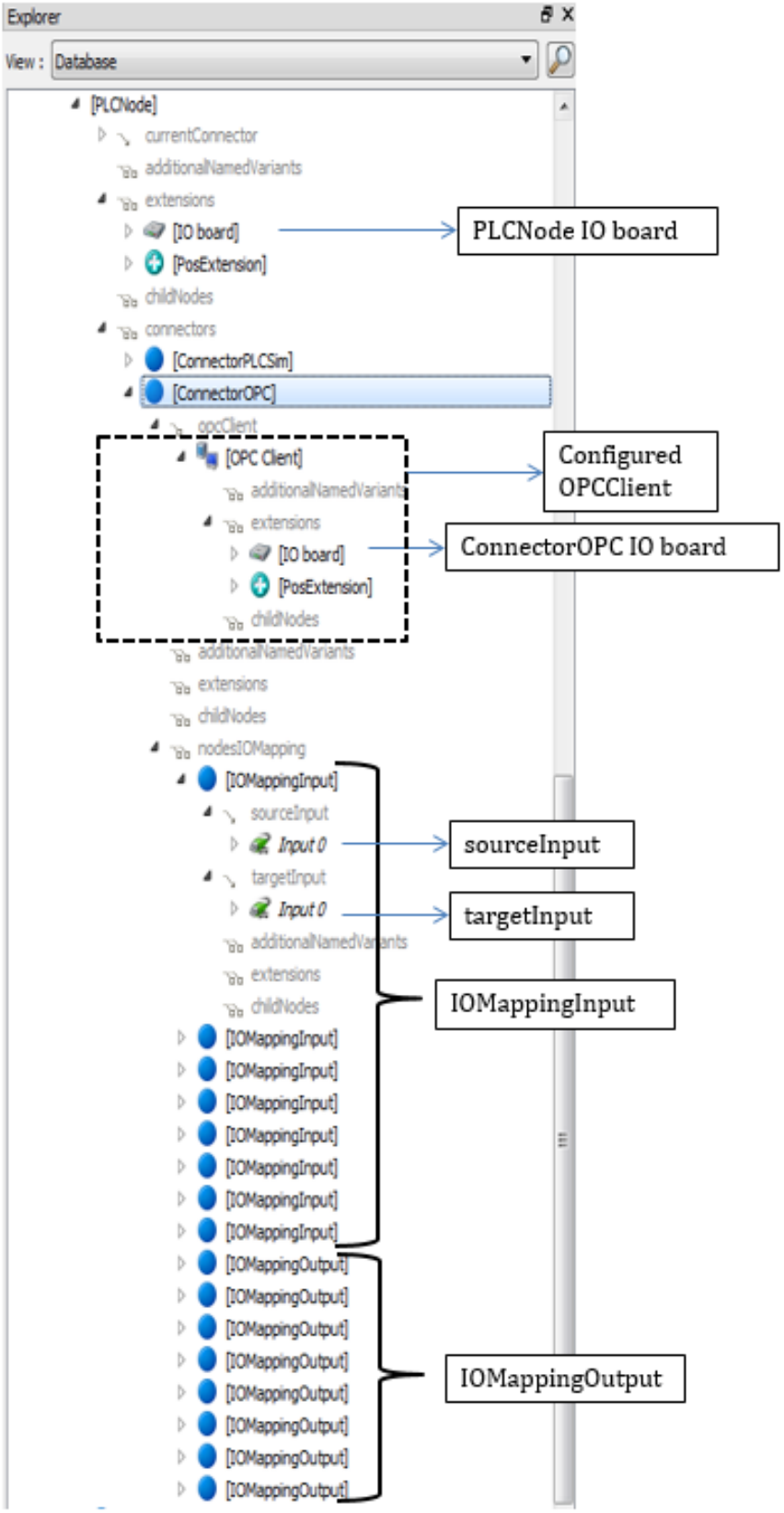


Figure 5.8: Configured ConnectorOPC

After both the connectors are configured to connect to external PLCs, the working of the model is tested with the PLC program on the external PLCs. At first the simulated PLC S7-PLCSIM is chosen as the external PLC to control the behavior of the simulation model. To connect the simulation model with S7-PLCSIM, the ConnectorPLCSim is selected as the CurrentConnector. Starting the simulation, the S7-PLCSIM is set to 'Run' mode, and the exchange of data between the PLCNode and S7-PLCSIM begins. When the simulation is started, the input signals are sent to the Simulated PLC and the program on the simulated PLC sends appropriate output signals. This communication of input and output signals continue as long as the simulation is running. The model is seen to behave as mentioned earlier in this chapter. The simultaneous change in input and output signals can be monitored in the S7-PLCSIM simulation window as well as in the PLCNode and ConnectorPLCSim IO board in VEROSIM IO editor.

The PLCNode's operation is validated by selecting the connection to simulated PLC and hardware PLC one at a time. Before starting the simulation, the user must ensure the following points are taken care of:

1 The number and order of inputs and outputs in the PLCNode matches the number and order of inputs and outputs defined in the PLC programs for simulated PLC S7-PLCSIM.
2 Inputs and outputs of the OPCClient IO board are mapped to the OPCItems served by the OPC server.
3 The targetInput and sourceOutput in the IOMappingInput and IOMappingOutput for the ConnectorOPC are assigned appropriately.
4 PLC programs are downloaded to the simulated and hardware PLC, and the hardware PLC S7-300 is set to 'run' mode.

Validation of the connection to external hardware PLC is done by selecting the ConnectorOPC as CurrentConnector. Starting the simulation connects the OPCClient in the ConnectorOPC to the OPC server. As ConnectorOPC is selected as currentConnector the IO maps of the ConnectorOPC are active i.e. the PLCNode IO board is mapped to the OPCClient IO board in the ConnectorOPC. The input signals from PLCNode are sent to the hardware PLC S7-300 via the OPCClient and OPC server. The program running on the S7-300 generates appropriate output signals which are communicated back to the PLCNode via OPC server and OPCClient object. On pressing the 'Start' button on the

screen the model begins the operation and it is seen that the model behaves in an expected manner. A test OPC client installed along with the software ‘Softing OPC server Ethernet’ can be used to configure the OPCItems of interest for monitoring. When the inputs and outputs change the state at the PLCNode, the same change in the state can be observed at the corresponding OPCItem in the test OPC Client.

For validation of the connection to external simulated PLC, the operation of the model and the simulation is stopped. The currentConnector is now changed to ConnectorPLCSim. Starting the simulation makes the IO maps for the ConnectorPLCSim active, i.e. the PLCNode IO board is mapped to the ConnectorPLCSim IO board. S7-PLCSIM is set to ‘run’ mode at the start of the simulation. The input signals from the PLCNode are sent to the simulated PLC S7-PLCSIM. The simulated PLC generates output signals by processing the inputs and the instructions stored in the program memory of the PLC. On pressing the ‘Start/Stop’ button on the screen the model begins the operation and it is seen that the model behaves in an expected manner on connecting to S7-PLCSIM as well. When the inputs and outputs change the state at the PLCNode, the same change in the state can be observed at the S7-PLCSIM simulation window as well.

6 Conclusion and Outlook

The modern day manufacturing industries involve highly automated production lines with many complex and interacting systems. Commissioning of these complex systems is a time consuming and expensive task. Virtual verification of production lines with the help of simulation models prior to actual commissioning helps in reducing the delays and additional costs due to errors during actual commissioning.

Including the behavior of control systems (PLCs) into a simulation environment helps in generating more realistic models. These models help in the verification of the complex system along with the PLC programs. As external PLCs can reproduce more accurate behavior of the control system, a generic concept is developed in this master thesis to integrate external PLCs into graphic simulation environment.

A concept of generic controller object in simulation environment is developed, which allows integration of different type of external PLCs into the simulation environment. The controller object has inputs and outputs to connect to the sensors and actuators in the simulation environment.

To facilitate the connection to different types of external PLC, the concept of connectors is introduced. Each of the connector has inputs and outputs which are mapped to the inputs and outputs of the connected external PLC via a communication interface respectively. On establishing the connection between the connector and the external PLC, the inputs of the connector are written to the inputs of the external PLC. The PLC generates appropriate output signal by processing the input signals and the instructions stored in the PLC's memory. The outputs of the connector are read from the output of the external PLC via the communication interface.

The controller object connects to the sensors and actuators in the simulation environment and the external PLC exchanges the inputs and outputs with the connector, an IO mapping function is responsible for mapping the inputs and outputs of the controller object to the inputs and outputs of the connector. This IO mapping function is executed every simulation cycle of the simulation software. Thus the input and output variables are updated every simulation cycle.

Connections to more than one external PLC can be established in a single controller object by configuring multiple connectors. However, to avoid the conflicts, connection to only one of the external PLCs can be active at a given time. All the configured connectors are listed in the controller object and the user can select one of the connectors to be active at a given time. The active connector maps its inputs and outputs to the inputs and outputs of the controller object, thus connecting the sensors and actuators to the external PLC.

The generic PLC integration concept is implemented using VEROSIM as simulation system. A generic controller object is developed in the VEROSIM environment which is capable of integrating different external PLCs.

As a prototype implementation connections to external simulated PLC S7-PLCSIM and hardware PLC S7-300 are implemented. This demonstrates the ability of the controller object to create connections to more than one external PLC. Two types of connectors can be created in the controller object to connect to the simulated and hardware PLC. The connection to the simulated PLC is via a COM object and the connection to the hardware PLC is via an OPC client-server configuration. A third party OPC software is used to configure the OPC server.

The controller object's operation is validated with the help of a mechatronic model in VEROSIM. Sensor and actuator signals from the simulation model are connected to the inputs and outputs of the controller object. The control programs necessary to govern the behavior of the mechatronic model are developed using a PLC program development tool and downloaded to the external simulated and real PLC.

Two connectors are configured in the controller object to integrate the external simulated and real PLC into the simulation environment. IO maps are created to map the inputs and outputs of each of the connector to the controller object. Behavior of the simulation model is observed by selecting one of the configured connectors as active at a given time.

Running the simulation in real-time the behavior of the model is observed. Selecting the connector to the simulated PLC as active connector, the model is seen to behave in an expected manner. The switching the active connector which connects to the hardware PLC, the model was seen to behave in the exact same manner. This demonstrates the functioning of the generic controller object in the simulation environment.

As a future development a connection to the hardware PLC can be implemented using libraries such as ‘libnodave’. This eliminates the requirement of additional OPC client-server software. Another improvement can be implementation of the IO mapping function using the signal-slot mechanism. In this thesis the IO mapping function is implemented such that it is executed every simulation cycle of the simulation software. Thus the input and output variables are updated every simulation cycle. Mapping the inputs and outputs every simulation cycle becomes redundant if the input and output variables are not changing at a greater speed. This redundancy can be overcome by implementing the IO mapping function using the signal-slot mechanism. Every change in the input or output variable generates a signal. This signal is connected to a slot which implements the IO mapping function. The concept of generic PLC integration presented in this thesis can be implemented using different graphic simulation system.

References

[1] B. K. Choi and B. H. Kim, "New Trends in CIM," in *Current Advances in Mechanical Design*, pp. 425–436, 2000.

[2] S. C. Park, C. M. Park, and G.-N. Wang, "A PLC programming environment based on a virtual plant," *The International Journal of Advanced Manufacturing Technology*, vol. 39, no. 11-12, pp. 1262–1270, 2008.

[3] G. Reinhart and G. Wünsch, "Economic application of virtual commissioning to mechatronic production systems," *Production Engineering*, vol. 1, no. 4, pp. 371–379, 2007.

[4] S. Makris, G. Michalos, and G. Chryssolouris, "Virtual Commissioning of an Assembly Cell with Cooperating Robots," *Advances in Decision Sciences*, vol. 2012, no. 1, pp. 1–11, 2012.

[5] J. Dzinic and C. Yao, *Simulation-based verification of PLC programs*: Department of Signals and Systems, Chalmers University of Technology, ISSN: 99-2747920-4, 2014.

[6] Z. Liu, C. Diedrich, and N. Suchold, *Virtual Commissioning of Automated Systems*: INTECH Open Access Publisher, 2012.

[7] *VDI-Richtlinie 4499, Digitale Fabrik Grundlagen, Blatt 1, Duesseldorf*, 2007.

[8] O. Salamon and A. Heidari, *Virtual commissioning of an existing manufacturing cell at Volvo Car Corporation using DELMIA V6*, no: EX023/2012: Department of Signals and Systems, Chalmers University of Technology, 2012.

[9] F. Auinger, M. Vorderwinkler, and G. Buchtela, "Interface driven domain-independent modeling architecture for "soft-commissioning" and "reality in the loop"," in *Farrington, Nembhard et al. (Ed.) – the 31st winter simulation conference*, pp. 798–805, 1999.

[10] *MATLAB.* Available: http://www.mathworks.com/products/simulink/ (2015, Aug. 02).

[11] P. A. Fritzson, *Principles of object-oriented modeling and simulation with Modelica 2.1*. Piscataway, N.J, [New York]: IEEE Press; Wiley-Interscience, 2004.

[12] Cosmol, *http://www.comsol.com/* (Accessed on: 2015, Aug. 10).

[13] J. Rossmann, M. Schluse, C. Schlette, R. Waspe, "A New Approach to 3D Simulation Technology as Enabling Technology for eROBOTICS," in *1st International Simulation Tools Conference & EXPO 2013, SIMEX'2013 (Van Impe, Logist, eds.),*, pp. 39-46,2013.

[14] http://wiki.ros.org/gazebo, *Gazebo* (Accessed on: 2015, Aug. 05).

[15] W. Bolton, *Programmable logic controllers*, 5th ed. Amsterdam, Boston: Newnes, 2009.

[16] Simmechanics from Mathworks, *http://www.mathworks.com/products/simmechanics/* (Accessed on: 2015, Aug. 01).

[17] K.-H. John and M. Tiegelkamp, *IEC 61131-3: Programming industrial automation systems: Concepts and programming languages, requirements for programming systems, decision-making aids*, 2nd ed. Berlin, New York: Springer, 2010.

[18] *S7-PLCSIM Manual.* Available: https://cache.industry.siemens.com/dl/files/828/54667828/att_110511/v1/s7wsvhdb_en-US.pdf (date of access: 2015, Aug. 02).

[19] *STEP-7 Manual.* Available: https://support.industry.siemens.com/cs/document/45531107/simatic-programming-with-step-7-v55?dti=0&lc=en-WW (date of access: 2015, Aug. 02).

[20] Li Zheng and H. Nakagawa, "OPC (OLE for process control) specification and its developments," in *SICE 2002. of the 41st 5-7*, pp. 917–920, 2002.

[21] M. R. Anwar, O. Anwar, S. F. Shamim, and A. A. Zahid, "Human Machine Interface Using OPC (OLE for Process Control)," in *Student Conference On Engineering, Sciences and Technology*, pp. 35–40, 2004.

[22] M. H. Schwarz and J. Boercsoek, "Advances of OPC Client Server Architectures for Maintenance Strategies-a Research and Development Area not only for

Industries," *WSEAS Transactions on Systems and Control*, vol. 3, no. 3, pp. 195–207, 2008.

[23] M. H. Schwarz and J. Boercsoek, "A survey on OLE for process control (OPC)," in *World Scientific and Engineering Academy and Society (WSEAS) – Proceedings of the 7th Conference*, pp. 186–191, 2007.

[24] Roßmann, J, Eilers, K, Schlette, C, Schluse, M, "A Uniform Framework to Program, Animate and Control Objects, Kinematics and Articulated Mechanisms in a Comprehensive Simulation System," in *Proceedings of the Joint 41th International*