People’s Democratic Republic of Algeria

Ministry of Higher Education and Scientific Research

University M’Hamed BOUGARA – Boumerdes

Institute of Electrical and Electronic Engineering

Department of Electronics

Final Year Project Report Presented in Partial Fulfilment of the Requirements for the Degree of

MASTER

In **Control**

Option: **Control**

Title:

Hardware-In-the-Loop Simulation for Product-driven Control of a Cyber-Physical Manufacturing System

Presented by:

- **HALLEDJ Salah Eddine**
- **MAHMOUD BACHA Anis**

Supervisor:

Dr. OUADI Abderrahmane

Registration Number:………/2020

Dedication

Above all, all the thanks and the praise be to our almighty god for his infinite generosity.
I dedicate this work to
The strongest and gentlest soul, my mother HADJI S. who thought me to trust in Allah, believe in hard work and that so much could be done with little.
To my father Ahmed, for earning an honest living for us and for supporting me to believe in myself.
This work is also dedicated to
My dear sisters Zakia, Samah and Amina and their husbands Moussa and Ahmed respectively.
All HALLEDJ and HADJI Families and my closest friends Isahk and Malak.
Special thanks to my friend and partner in this work Mr. MAHMOUD BACHA Anis.
We cannot express enough thanks to my educational family, teachers and classmates for their advices, technical guidance.

HALLEDJ S.E.

At the very outset, all our prayers and thankfulness are to Allah the almighty for facilitating this challenge.
I dedicate this work to my dear parents Mohamed and SLIMANI F. whose efforts can never be fairly rewarded, I wish at least they can see a fruit of their sacrifices in my success.
This work is also dedicated to my brother Samy, my sister Hiba, all MAHMOUD BACHA and SLIMANI families, all closest and farthest friends and those dear irreplaceable people no matter how far they are.
Special thanks to my amazing partner in this project Mr. HALLEDJ Salah Eddine and his family.
No dedication would be complete without expressing my appreciation and thankfulness to all teachers I had during all educational phases.

MAHMOUD BACHA A.

Acknowledgement

This project was proposed by Robotic Production Systems (SRP) division of the Center for the Development of Advanced Technologies (CDTA) in Baba Hassen.

This work would not come to fruition without the support of several individuals and organizations to which we will be grateful forever.

We wish to express our everlasting gratitude to our professor and supervisor, Dr. OUADI Abderrahmane, for his enthusiasm, patient and helpful information that were crucial through the study as well as the work.

We cannot express enough thanks to our co-supervisor and representative of the Center for the Development of Advanced Technologies (CDTA), Mr. MIHOUBI Bachir who gave us the golden opportunity to work on this wonderful project on the topic "Hardware in the loop simulation for Product driven control of a cyber-physical manufacturing system", and also helped us with his experience and professional expertise in simulation, artificial intelligent and manufacturing management.

We would like to thank the Institute of Electrical and Electronic Engineering (IGEE) for amazing five years of study and all the facilities that we found inside the institute.

Abstract

The Product driven system (PDS) is considered as distributed control system (DCS) in which the product plays a major role in decision-making. This project focuses on how to develop a cost-effective validation strategy of PDS applied to the highly automated flexible robotized assembly system based on hardware-in-the-loop simulation approach (HILS). An efficient Cyber-physical system (CPS) is developed for a discrete flexible manufacturing system. Mainly, this proposed CPS employs a dynamic multi-agent scheduling mechanism (MAS) based on priority dispatching rules to manage, control the 3D discrete event simulation (DES) and initiate PDS paradigm. In the other side, Radio Frequency Identification (RFID) - as auto-identity - allows an effective real-time tracking and progress monitoring of each product across its lifecycle in the shop floor, the controller (PLC, Arduino) is the heart of HILS approach which synchronizes the cyber space (decisional point) with the physical one.

Keywords: hardware-in-the-loop simulation (HILS), discrete event simulation (DES), product-driven system (PDS), multi agent system (MAS), radio frequency identification (RFID), cyber-physical system (CPS) , flexible manufacturing system (FMS).

N.B. This project was planned to be verified in an industrial scale environment using real Siemens PLCs and RFID systems as well as an industrial turntable. Unfortunately, due to the pandemic and its repercussions, the training was cancelled and the industrial hardware was not accessible. The resort to a simulated PLC was then unavoidable. The insertion of small scale electronic components (hardware part) such as: Arduino microcontroller and RC522 RFID module was in order to maintain HILS as validation strategy. A small DC motorized turntable was implemented too.

Contents

List of Abbreviations

μC Microcontroller.

A&E Alarms and Events.

ACL Agent Communication Language.

AgM Machine agent.

AgP Product agent.

AgR Routing agent.

ALU Arithmetic and logic unit.

AMS Agent Management System.

APDU Application layer Protocol Data Unit.

ASCII American Standard Code for Information Interchange.

CPU Central processing unit.

DA Data Access.

DCOM/COM Distributed Component Object Model.

DCS Distributed Control System.

DES Discrete Event Simulation.

DF Directory Facilator.

EPROM Erasable and programmable read-only-memory.

FBD Functional Block Diagram.

FIFO First Input First Output.

FMS	Flexible manufacturing system.
HDA	Historical Data Access.
HILS	Hardware-In-the-Loop Simulation.
HTTP	HyperText Transfer Protocol.
IEC	International Electrotechnical Commission.
IL	Instruction List.
IP	Internet Protocol.
JADE	Java Agent DEvelopment Framework.
LAN	Local Area Network.
LD	Ladder Diagram.
LIFO	Last Input First Output.
MAC	Media Access Control.
MAS	Multi-agent system.
MISO	Master In Slave Out.
MOSI	Master Out Slave In.
OPC	Open Platform Communication.
PDRs	Priority dispatching rules.
PDS	Product-driven system.
PLC	Programmable logic controller.
RAM	Random-access memory.
RF	Radio frequency.
RFID	Radio frequency identification.
RO	Read-only.
ROM	Read-only memory.

RTU	Remote Terminal Unit.
RW	Read/write.
SCK	Serial Clock.
SFC	Sequential Function Chart.
SILS	Software-In-the-Loop Simulation.
SOAP	Simple Object Access Protocol.
SPI	Serial Periferal Interface.
SS	Slave Select.
TCP	Transmission Control Protocol.
UA	Unified Architecture.
XML	eXtensible Markup Language.

List of Figures

List of Tables

General Introduction

Industrial automation systems continuously get more complex and growing over time due to the global competition that is characterized by diversification of production and increasingly shorter innovation cycles in addition to the appearance of smart technologies and instruments (next generation instruments). In recent decades, a new industrial era is beginning to switch from classical automation systems to the smart automation factory, opening up the so-called "Industry 4.0" or the fourth industrial generation[1, 2].

Accordingly, "*modern manufacturing is experiencing a paradigm shift towards more flexibility and reconfigurability (physically and logically) to respond quickly and efficiently to changing production requirements and market demands"* (ElMaraghy et al.,2011). Physical reconfiguration means hardware changes e.g. plant layout or machinery. Logical reconfigurations are software changes; especially major changes of control software frequently caused by the hardware changes which so-called cyber- physical systems CPS. They are defined as an association between cyber computational space (information feedback) that represents the intelligent data management, analytics and computational capabilities (artificial intelligent, machine learning, 3D simulation, communication protocols as OPC UA, Modbus) and physical space that represents the manufacturing components (next generation sensors (RFID), PLC, microcontroller, etc.) [3, 4].

The cyber-physical platform will include a dynamic multi-agent scheduling mechanism based on priority dispatching rules assigned to different products and machines. Besides, realistic scheduling constraints and a finite capacity queue of machines are considered in the scheduling problem.

In addition, the integration of RFID technology with production scheduling can significantly improve scheduling performance and productivity [5, 6].

Furthermore, HILS is used for real-time monitoring and controlling of products and production lines [7]. Benefiting from communication technologies, real-time data synchronization is possible now by real-time communication between the digital and the physical equipment.

The aim of this project is to develop and implement PDS (intelligent product)[8] within CPS, to achieve the purpose; it needs to highlight the potential combination of RFID, MAS, 3D discrete simulation and HILS approach to substantiate the practical feasibility of the PDS paradigm within a realistic production environment. Providing a framework for synchronizing the physical flow of products and the associated information flow.

All these approaches are employed to verify and validate routing decisions of manufacturing control that enable real-time scheduling , furthermore, the proposed strategy has the benefits of reduced hardware integration and test costs while maintaining the reliability and maintainability of the proposed distributed control system (DCS) (see chapter 4).

Report Organization

This report is made up of four chapters, the first three chapters concern bibliographical study on various concepts in this work as well as the different devices and electronic components used.

Thus, the first chapter includes generalities on simulation and some information about Flexsim software.

The second chapter consists about all techniques and protocols used to link the cyberspace and hardware space.

In the third chapter, it describes the physical devices that are used to close the loop of HILS model as RFID, PLC, Arduino uno and Ethernet shield.

Finally, the last chapter focuses on the implementation of the cyberspace (MAS, DES model) and physical space as well as representing the main results.

Chapter 1

Introduction to Simulation with FlexSim Software

Introduction

Scientists, engineers, and practitioners of many professions have long relied on the creation of models to understand the studied phenomena. Traditional mathematical methods (i.e. differential equations) have been used for centuries as the main tool for analysis, comprehension, design, and prediction for complex systems in varied areas. However, these methods appeared unsuitable for studying the complex human-made systems developed during the twentieth century.

The emergence of digital computers provided alternative methods of design and analysis for both natural and artificial systems. Since the early days of computing, users translated their analytical models into computer-based ones. This approach allowed solving problems with a level of complexity unknown in earlier stages of scientific development.

Computer-simulated models also have additional benefits. They can be executed safely, experiments can be easily repeated in a cost effective and risk-free environment; thus, they are well suited for training purposes.

This chapter focuses on the concept of modeling and simulation theories especially on Discrete Event Simulation (DES) designed via FlexSim software.

1.1 Model and Simulation

A model is an entity that is used to represent some other entity for some defined purpose. In general, models are simplified abstractions, which embrace only the scope and level of detail needed to satisfy specific study objectives. Models are employed when investigation of the actual system is impractical or prohibitive. Indeed, models can be used to study systems that exist only in concept.

Simulation is a particular approach to studying models, which is fundamentally experiential or experimental. In principle, simulation is much like running field tests, except that the system of interest is replaced by a physical or computational model.In many applications, simulation also involves testing and comparing alternative designs and validating, explaining, and supporting simulation outcomes and study recommendations.[9]

1.2 Application domain

We might divide applications of simulation broadly into two categories. The first includes so-called man-in-the-loop simulations used for training and/or entertainment. Many professionals hone their skills and learn emergency procedures in simulated environments, which are safe from the consequences of inexperience and failure. Pilots train in flight simulators in order to experience the cockpit of a particular aircraft, nuclear power-plant operators routinely recertify in control-room simulators and physicians learn new procedures employing simulated patients.

The second category includes the analysis and design of artifacts and processes. This is the technical domain, which engineers and operations researchers most commonly associate with simulation. Consider for example the design of a new aircraft or managing manufacturing systems.

1.3 DES model

DES is a computer modeling where the state of a system is represented by a chronological sequence of events.(i.e. it models the operation of a system as discrete of events in time) . Each event occurs at a particular instant in time and marks a change of state in the system. Between consecutive events, no change in the system is assumed to occur; thus, the simulation time can directly jump to the occurrence time of the next event, which is called next-event time progression [10], as shown in Figure 1.1 .

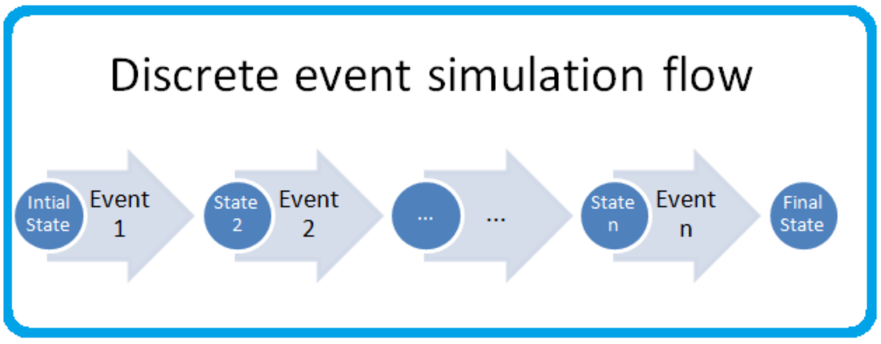


Figure 1.1: Discrete event simulation steps

1.4 FlexSim simulation software

1.4.1 FlexSim

It is a powerful and easy-to-use modeling and simulation software tool that allows the user to construct 3D computer simulation model of a real-life system and run experiments on the model. FlexSim is a discrete-event simulation software tool that provides realistic graphical animation and extensive performance reports that enables the user to identify problems and evaluate alternative solutions in a short amount of time [9]. This is why we used in this project for its flexibility and abundance of its library, which made it easy for us to create important communication networks via sockets. [11]

1.4.2 History

FlexSim was founded in 1993 by Bill Nordgren (Co-Founder Promodel Corporation, 1988), Roger Hullinger, and Cliff King, originally under the name F&H Simulations, Inc. F&H Simulations sold, supported, and conducted training courses for Taylor II simulation software owned and developed by Holland's FH Simulation B.V (F&H Holland).

In 1998, F&H Holland developed the first generation 3D object oriented simulation engine Taylor ED (Enterprise Dynamics). F&H Simulations assisted with the development of robust objects for use in Taylor ED.

In 2000, F&H Holland was acquired, F&H Simulations became independent, and a new simulation product started development under the guidance of Dr.Eamonn Lavery and lead programmer Anthony Johnson. This new 3D simulation product, known now as FlexSim, coincided with F&H Simulations.[12]

1.4.3 3D Object Library

The 3D objects in the FlexSim Library are the basic building blocks that uses to build a 3D model. Each object has built-in logic that is commonly used in a variety of simulation models. We can also easily edit the properties and customize the logic on these objects to adapt any of them to the unique needs of the simulation project.[13]
Hence, the common objects and categories are represented in the same order as they appear in the library:

(a) **Fixed resource**:
Fixed resources are objects that remain fixed or stationary in the model. They interact with flow items in the simulation, such as storing or modifying flow items, which are shown in Table 1.1

Name Icon	**Description**
Source	The source creates flow items and releases them to a downstream object. We can control the rate at which the source creates flow items so that they arrive on a fixed schedule, a regular continuous rate, or a random statistical distribution.
Queue	The queue stores flow items until a downstream object is ready to take them. By default, the queue releases flow items on a first-in-first-out basis, but other options are available.
Processor	Processors simulate flow items getting processed at a station. Processors simulate a time delay, beginning with a setup time followed by the process time.
Combiner	The combiner groups multiple flow items together. It can either join the flow items together permanently, or it can pack them into a container flow item so that they can be separated at a later point in time.
Separator	It separates a flow item into multiple parts, either by unpacking a container flow item that has been packed by a combiner or by making multiple copies of the original flow item
Multiprocessor	The multiprocessor is similar to the processor object, but it can simulate flow items going through a sequence of two or more processes. You can require the multiprocessor to use an operator in any or all of these processes.

Table 1.1: Fixed source elements

(b) **Task executers**:

Task executers are objects that can move throughout the model and interact with fixed resources and flow items. These objects can travel, load flow items, unload flow items, act as shared resources for processing stations, and perform many other simulation tasks. All task executers have the same basic functionality; the main difference between them is the way they move (see Table 1.2).

Name Icon	**Description**
Operator	Operators represent employees that can transport flow items, operate fixed resources, and perform a variety of other tasks that require an employee in a simulation model
Robot	The robot is special transport that lifts flow items from their starting locations and places them at their ending locations.
Dispatcher	The dispatcher is used to control a group of transporters or operators. Fixed resources can send task sequences to the dispatcher, which then delegates the tasks to the transports or operators that are connected to it once they become available.
Elevator	The elevator is a special type of transport that moves flow items up and down. It will automatically travel to the level where flow items need to be picked up or dropped off.

Table 1.2: Task executor elements

(c) **Conveyors:**

Table 1.3 will provide a high-level overview of these objects:

Name Icon	**Description**
Straight Conveyor	The straight conveyor can simulate conveyor belts or roller conveyors.
Curved Conveyor	This conveyor has a curved shape with varying radius settings.
Join Conveyors	Join conveyors acts more like a tool than an object. Use it to create a curved conveyor connecting two conveyor sections.

Table 1.3: Conveyor types

(d) **Programming language**

Flexsim is developed in C++ programming language using Open GL technology allowing users greater flexibility to develop models to match their own processes. Each Flexsim license includes a copy of Visual C++.net and ExpertFit distribution module.[11]

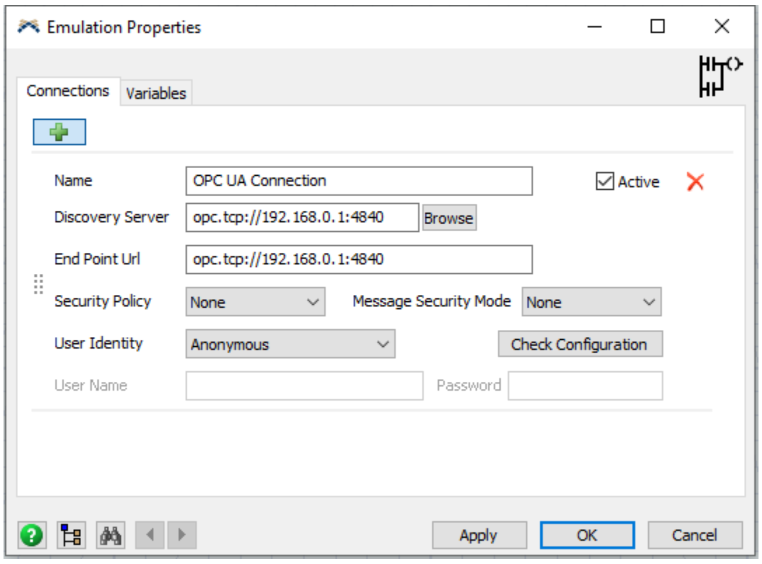
1.4.4 Emulation Tool

The Emulation tool creates a link between FlexSim and external PLCs or clients/servers that communicate with PLCs. This tool can create multiple connections and define variables for each of those connections. This tool supports three protocols: OPC UA, OPC DA and Modbus [14].

OPC UA protocol was recently adopted by Flexsim in his 2020 version, and it is the one we are going to use in this study to link Flexsim to a PLC; hence, it is the method we should focus on in this description.

1.4.4.1 OPC UA as an emulation tool in FlexSim

Figure 1.2 is the emulation properties window and comprises two tabs: connections and variables.



Name: OPC UA Connection ☑ Active
Discovery Server: opc.tcp://192.168.0.1:4840
End Point Url: opc.tcp://192.168.0.1:4840
Security Policy: None
Message Security Mode: None
User Identity: Anonymous
User Name
Password

Figure 1.2: Emulation via OPC UA configuration

Connection tab contains data and parameters to establish a connection to an OPC UA server.

- **Name:** The name of the connection. Used for identifying the connection in FlexSim.

- **Active:** If checked, a connection will be created to connect FlexSim to an OPC server. This occurs when the model starts running.
- **Discovery Server:** The ip address of the discovery server.
- **End Point URL:** The specific URL defining the server and protocol to connect to. We can auto-fill this field using the Browse button.
- **Security Policy:** Defines the set of security algorithms and key length to use when communicating with the server. [14]

Variables tab (Figure 1.3) contains imported tags from OPC UA server and their information. OPC UA variable in Flexsim can be either a *Sensor* or a *Control*. Sensors write values to the tag once a preselected *event* occurs; whereas, controls read tag values and execute a predefined *action* depending on those values.

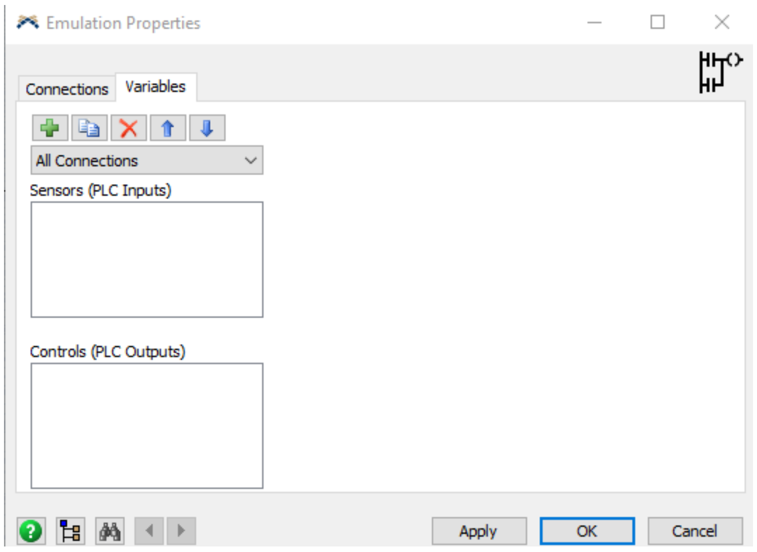


Figure 1.3: OPC UA variables tab

Conclusion

This chapter contains general information on the simulation, the type of simulation used, and a global view on the Flexsim software as well as the different 3D objects that make up its library.

Chapter 2

Communication Solutions for Industrial Applications

Introduction

This chapter presents theories of techniques we use to link Jade multi agent system with the hardware (PLC, DCS, HMI and Arduino) indirectly to comprise into a distributed industrial control system based on agents, using several communication protocols.

Industrial applications of MAS are limited, among others, especially due to the difficulties of communication between agents' development environments and heterogeneous set of control devices, sensors and actuators that can be found in an industrial process.

The solution involved the use of TCP/IP socket between MAS written in Java and the virtual devices in the simulator created in FlexSim. Utilizing OPC UA which allow access to process variable between the emulator and hardware, what makes a synchronization between the physical world (controller as PLC, Arduino) and the cyber world (emulator).

This chapter focuses on the theories and concepts of developing MAS as well as the communication solution protocols as OPC standard, TCP/IP socket network and MODBUS protocol.

2.1 JADE

It is a software framework for developing MAS interoperable, used by a very heterogeneous community of users as a tool both for research and for development of industrial and commercial applications.[15, 16]

2.1.1 History of JADE

JADE was initially developed by Telecom Italia Lab. This sector is the R&D branch of Telecom Italia Group, which is responsible for promoting technological innovation. Telecom Italia conceived and promoted JADE by basing it in 2000. In March 2003, Motorola and Telecom Italia created the JADE Governing Board with the objective of promoting the development and adoption of JADE in the mobile telecommunications industry as middleware based. The JADE Governing Board accepts to any company and/or organization interested in the commercial use and exploitation of JADE to commit to its development and promotion.[17]

2.1.2 Platform

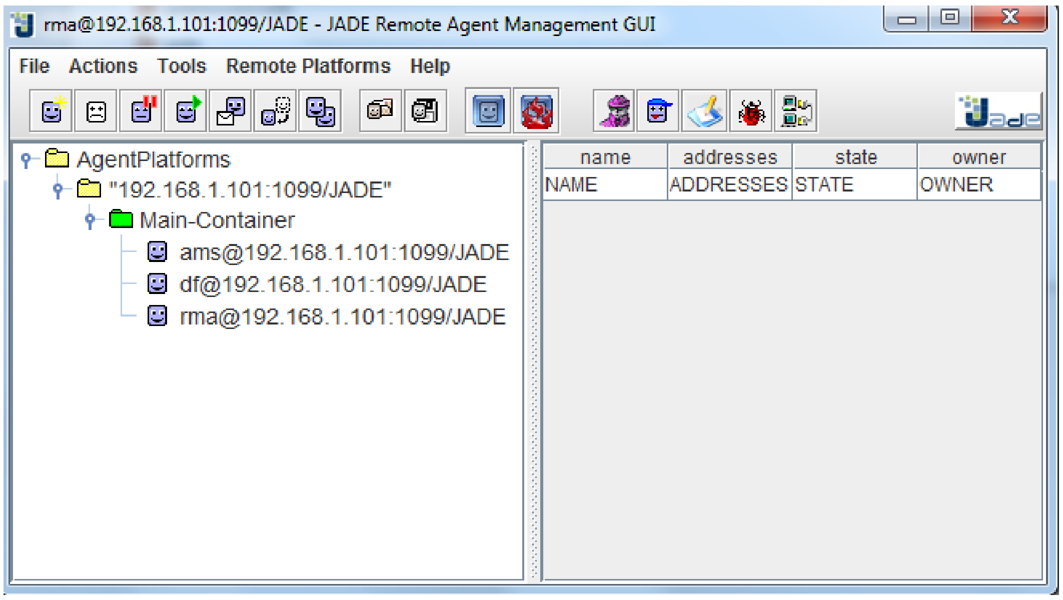
JADE is a distributed agents platform, which has a container for each host where we are running the agents (see Figure 2.1). Additionally, the platform has various debugging tools, mobility of code and content agents, the possibility of parallel execution of the behavior of agents, as well as support for the definition of languages. Each platform must have a parent container that has two special agents called AMS and DF.

2.1.2.1 DF Agent

The DF (Directory Facilitator) provides a directory, which announces which agents are available on the platform.

2.1.2.2 AMS agent

The AMS (Agent Management System) controls the platform. It is the only one who can create and destroy other agents, destroy containers and stop the platform.


rma@192.168.1.101:1099/JADE - JADE Remote Agent Management GUI
File
Actions
Tools
Remote Platforms
Help

AgentPlatforms
"192.168.1.101:1099/JADE"
Main-Container
ams@192.168.1.101:1099/JADE
df@192.168.1.101:1099/JADE
rma@192.168.1.101:1099/JADE

name	addresses	state	owner
NAME	ADDRESSES	STATE	OWNER

Figure 2.1: JADE-GUI platform

2.1.3 JADE agent

The cycle of life of a JADE agent follows the cycle proposed by FIPA. These agents go through different states defined as:

1. Initiated: The agent has been created but has not registered yet the AMS.
2. Active: The agent has been registered and has a name. In this state, it can communicate with other agents.
3. Suspended: The agent is stopped because its thread is suspended.
4. Waiting: The agent is blocked waiting for an event.
5. Deleted: The agent has finished and his thread ended his execute and there is not any more in the AMS.
6. Transit: The agent is moving to a new location.

2.1.4 Agents' behaviour

The behavior defines the actions under a given event. This behavior of the agent is defined in the method setup using the method addBehaviour. The different behaviors that the agent will adopt are defined from the abstract class Behaviour. The class Behaviour contains the abstract methods:

- $action()$: Is executed when the action takes place.

- $done()$: Is executed at the end of the performance.

We can override the methods $onStart()$ and $OnEnd()$ property. Additionally, there are other methods such as $block()$ and $restart()$ used for modifying the agent's behavior. When an agent is locked it can be unlocked in different ways. Otherwise we user can override the methods $onStart()$ and $onEnd()$ the agent possess.

2.1.5 ACL messages

Agent Communication Language is the base of communication between agents. Sending messages is done by the method send of the class Agent. In this method, we have to pass an object of type ACLMessage that contains the recipient information, language, coding and content of the message. These messages are sent asynchronously, while messages are received they will be stored in a message queue as shown in figure 2.2. There are two types of receiving ACL messages, blocking or non-blocking. For this provide methods $blockingReceive()$ and $receive()$ respectively. In both methods, we can make filtering messages to be retrieved from the queue by setting different templates.[18]

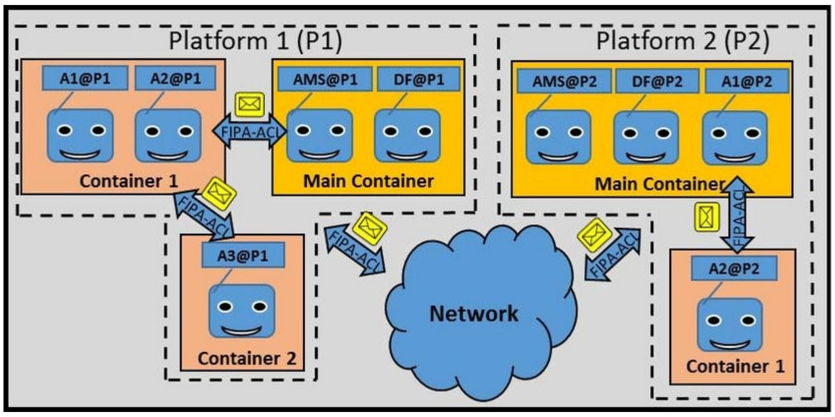


Figure 2.2: JAVA agent development framework (JADE) architecture [19]

2.2 Socket Communication

A socket is a one end-point that makes communication between two programs which are running on the same machine or over network. Socket are used to represent the connectivity between client and server. Socket is bound to an IP address and port number so that the TCP layer can identify the application.

Normally, a server runs on a specific computer and has socket which is bound to specific port number. The server waits the client to listen to the socket and makes a connection request. On the client end the client knows the hostname or IP address of the server and the port number of the server listening. Making a connection request the client program tries to negotiate with the server program on the hostname IP address and port number. When connection is establish between server and client, Client used that socket to communicate with server (read/write).[20]

Figure 2.3 describes how the server and client negotiates with each other on TCP layer over (specific IP address) port number.

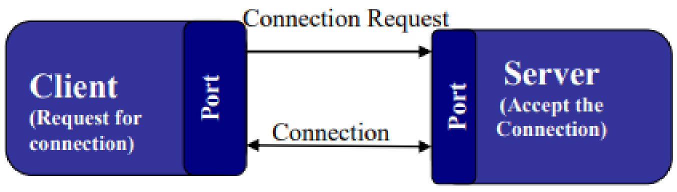


Figure 2.3: Connection between client and server on specific port

2.2.1 Java Socket

Network process in java uses sockets for two-way communications running on two different machines under the same underline IP address. A socket is an end point of two way communication between two different proxies running on two different networks.

Client actually initiates the process with the server process. Server process will bind to particular port number and wait for incoming client connections.

In Java, it is implemented with the socket end point for the server process using a class called ”*serversocket*” which is from ”*java.net*” package. In the client process, the class called ”*socket*” also from the ”*java.net*” package.

Once server socket is created, the server process can call ”*accept*” method to wait for incoming connections. To connect the client process, simply create a new socket and specify what IP address and port, the server socket is blocked down to accept for. Once we create that client socket, the server unblock the client process and communicate together (read/write). Blocking ”*accept*” for client side to make its socket instance, then we have the synchronization point on both. After that either one or both process close the socket (Figure 2.4).

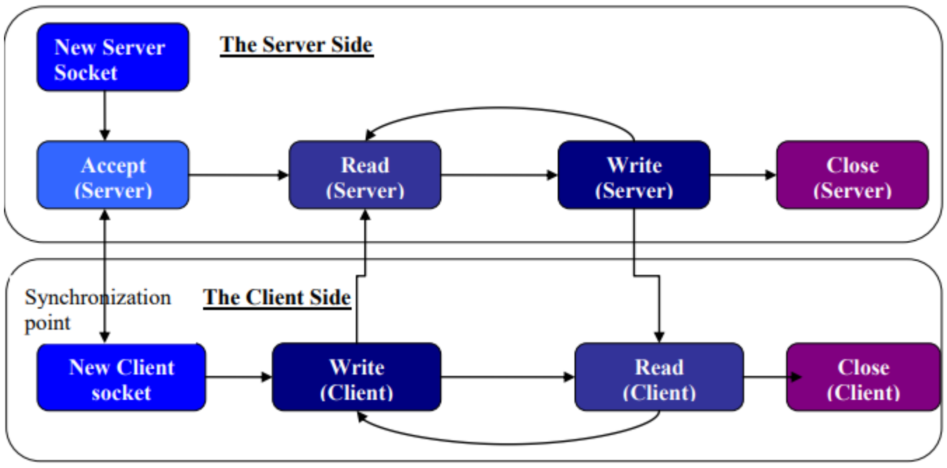


Figure 2.4: Overview of Java Sockets

2.2.2 TCP/IP Layer of Socket Communication

Transmission control protocol is a connection-oriented protocol. In this protocol, the connection between the sockets must be established. All data should be read immediately, as received. One listening zone called Server and other socket that asks to establish the connection called client. Server socket used to accept command of the client to use the open command. If the connection is established between them, then they communicate with each other. Socket creates new connection with its endpoint. It use listen/read for willing to accept the connection, send/write is send or write over the connection after the connection established and end used for release the connection.

Figure 2.5 shows the socket process on layers. After writing the code for Socket, which code works on presentation layer, the application layer does not know anything how socket works. Sockets reside on the Session Layer. The Session Layer is sandwiched between the application-oriented upper layers and the real-time data communication lower layers. TCP/IP maps the two layers. Computer transmits the data in the form of 0s and 1s.[21]

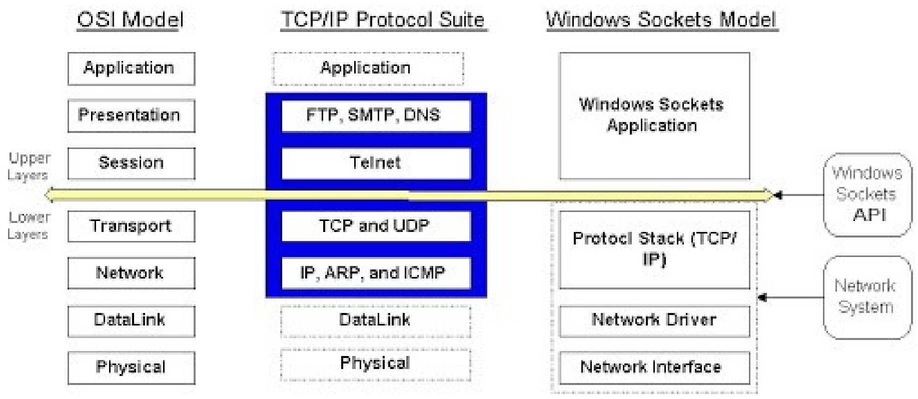


Figure 2.5: OSI layers (sockets, TCP/IP) [22]

2.2.3 Pseudocode of socket programming to link FlexSim and JADE

2.2.3.1 FlexSim part (server)

1. *Initialization of the socket:*
   Initialize the socket before trying to create a connection. It returns TRUE, if the initialization is successful and returns FALSE if it is not successful.

   ```
   socketinit();
   ```

   Or,

   ```
   if(socketinit()){ //statement
   }
   ```

2. *Creation of server socket:*
   Create a server socket using given port that will listen for a connection:

   ```
   servercreatemain(port_number);
   ```

3. *Accept the connection:*
   Accept command are used for accept the TCP connection from other system. The server application used the accept command to accept connection.

   ```
   client = serveraccept(0);
   //where index zero is the command blocks until a
   //connection is made. It returns an index used to
   //reference the connection made.
   ```

4. *Read from client :*
   Read the socket is used for get the data from another application on same system or another system using TCP Socket.

   ```
   input = serverreceive(int connection, char *buffer, int
       bufsize, int noblocking);
   ```

5. *Close the socket :*
   Close socket are used to release the connection after completing all receiving a sending data.

   ```
   servercloseconnection(getnodenum(client));
   //Close the connection to the client.
   socketend();//Stop using Windows sockets.
   setnodenum(client,0);//Set the client information to 0  to
       //indicate there is no longer a connection to the client.
   return 1;//Return a 1 to indicate the connection was closed.
   ```

2.2.3.2 JADE part (client):

There are some operations to be performed by a socket: open, accept connection, send data, receive data and close connection. ”*java.net*” package provides two classes, one is ”*socket*” which implement the client side of connection and the other one is ”*serversocket*”, which implements the server side of connection:

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.InetAddress;
import java.net.Socket;

Socket socket;
BufferedReader in;
PrintWriter out;
```

The operations are:

1. *Open Socket:*
   On client side create an object of Socket class. machine name/IP address and Port Number on which the server want to connect.
   ```
   socket = new Socket(InetAddress.getByName("address"),port
       number);
   ```
2. *Create Input Stream:*
   On client side *”DataInputStream”* class are use for receive the response for the server side. This class ”DataInputStream” allows to read the lines of texts. It has many methods *read*, *readChar*, *readInt*, *readDouble*, and *readLine*:
   ```
   in = new BufferedReader(new InputStreamReader(
       sock.getInputStream()));
   ```
3. *Close the socket:*
   On Client side:
   ```
   //close Input and Output Stream before closing the socket
   socket.close();
   ```

2.3 OPC standards

Open Platform Communication (OPC) is a series of standards and specifications for industrial telecommunication that specifies an interface between client applications for data processing and the servers that connect the physical industrial devices for control as PLCs, sensors and actuators (OPC Data Access/DA).

After the initial release in 1996, the OPC foundation was created to maintain the standard. As OPC has been adopted beyond the field of process control, the OPC foundation changed the name to Open Platform Communications in 2011. The change in name reflects the applications of OPC technology for applications in building automation, discrete manufacturing, process control and many others.[23, 24]

2.3.1 Client-server approach for information exchanche

An OPC server encapsulates the source of process information like a device and makes the information available via its interface. An OPC client connects to the OPC server and can access and consume the offered data. Applications consuming and providing data can be both client and server. The following figure shows a typical use case of OPC clients and servers. Classic OPC interfaces are based on the COM and DCOM technology from Microsoft (see Figure 2.6). Whereas, OPC UA (Unified Architecture) is an entirely new set of standards that incorporates all of the functionality of the classic standard (and more), but does so using cross platform web services and other modern technologies.

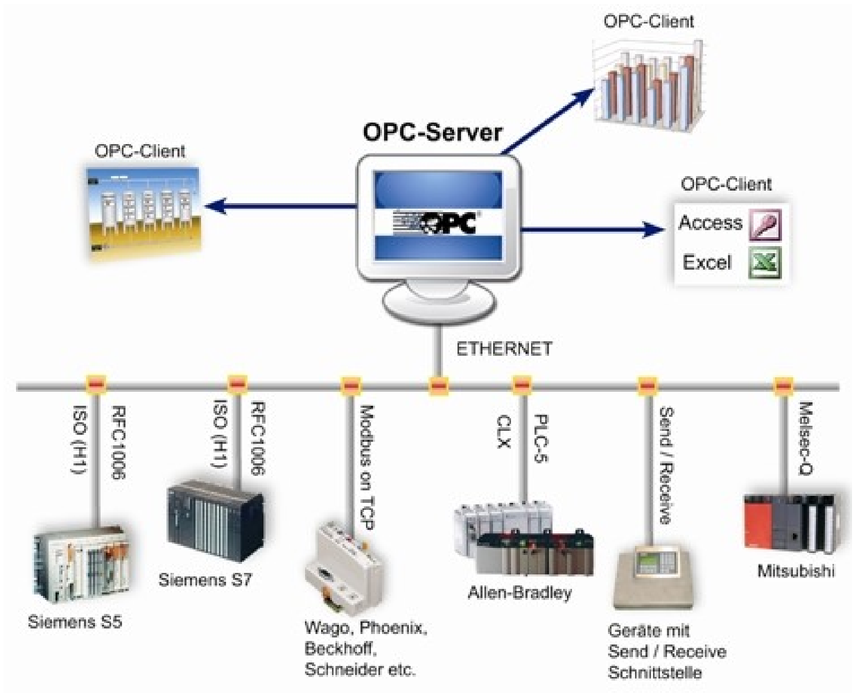


Figure 2.6: OPC server communication [25]

2.3.2 OPC specification groups

2.3.2.1 OPC DA

The OPC Data Access interface enables reading, writing, and monitoring of variables containing current process data. The main use case is to move real-time data from PLCs, DCSs, and other control devices to HMIs and other display clients. OPC DA is the most important OPC interface. It is implemented in 90 of the products using OPC technology today. Other OPC interfaces are mostly implemented in addition to DA.

OPC DA clients explicitly select the variables (OPC items) they want to read, write, or monitor in the server. The OPC client establishes a connection to the server by creating an OPCServer object. The server object offers methods to navigate through the address space hierarchy to find items and their properties like data type and access rights. For accessing the data, the client groups the OPC items with identical settings such as update time in an OPCGroup object. Figure 2.7 shows the different objects the OPC client creates in the server.

When added to a group, items can be read or written by the client. However, the preferred way for the cyclic reading of data by the client is monitoring the value changes in the server. The client defines an update rate on the group containing the items of interest. The update rate is used in the server to cyclic check the values for changes. After each cycle, the server sends only the changed values to the client. OPC provides real-time data that may not permanently be accessible, for example, when the communication to a device gets temporarily interrupted. The Classic OPC technology handles this issue by providing timestamp and quality for the delivered data. The quality specifies if the data is accurate (good), not available (bad), or unknown (uncertain).

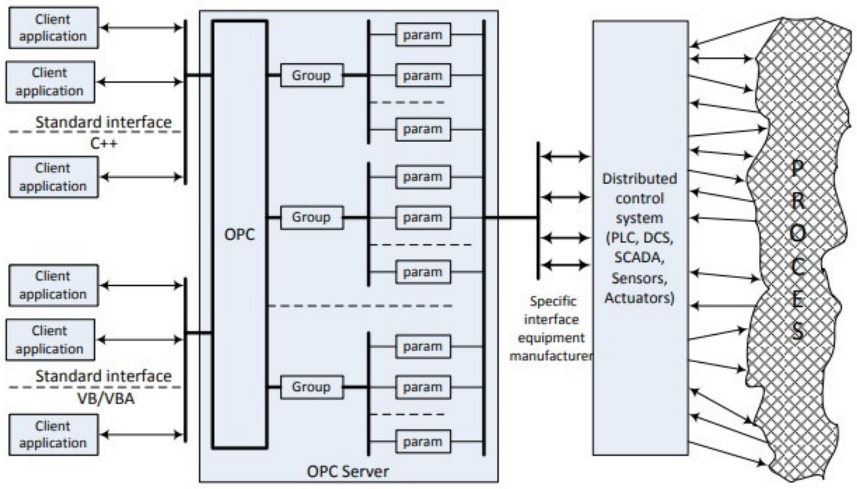


Figure 2.7: Structure of a control system using OPC server [26]

2.3.2.2 OPC A&E

The OPC A&E interface enables the reception of event notifications and alarm notifications. Events are single notifications informing the client about the occurrence of an event. Alarms are notifications that inform the client about the change of a condition in the process. Such a condition can be the level of a tank. In this example, a condition change can occur when a maximum level is exceeded or is fallen below a minimum level. Many alarms include the requirement that the alarm has to be acknowledged. This acknowledgement is also possible via the OPC A&E interface.

To receive notifications, the OPC A&E client connects to the server, subscribes for notifications, and than receives all notifications triggered in the server. To limit the number of notifications, the OPC client can specify certain filter criteria.

2.3.2.3 OPC HDA

Where OPC Data Access gives access to real-time, continually changing data, OPC Historical Data Access provides access to data already stored. From a simple serial data logging system to a complex SCADA system, historical archives can be retrieved in a uniform manner.

The OPC client connects by creating an OPCHDA Server object in the HDA server. This object offers all interfaces and methods to read and update historical data. A second object OPCHDA Browser is defined for browsing the address space of the HDA server.

In addition to the read methods, OPC HDA also defines methods for inserting, replacing, and deleting data in the history database.

2.3.2.4 OPC interface standards

OPC specified several additional standards as base specifications or for specialized needs. Base specifications are OPC Overview and OPC Common defining interfaces and behavior that is common to all COM-based OPC specifications. Figure 2.8 gives an overview for all Classic OPC specifications.

OPC Security specifies how to control client access to servers to protect sensitive information and to guard against unauthorized modification of process parameters.

OPC Complex Data, OPC Batch, and OPC Data eXchange (DX) are extensions to OPC DA. Complex Data defines how to describe and transport values with complex structured data types. OPC DX specifies the data exchange between Data Access servers by defining the client behavior and the configuration interfaces for the client inside a server. OPC Batch extends DA for the specialized needs of batch processes.

2.3.2.5 XML-DA

OPC XML-DA was the first platform-independent OPC specification replacing COM/DCOM with HTTP/SOAP and Web Service technologies. Thus a vendor and platform-neutral communication infrastructure was introduced and widely accepted functionality of OPC Data Access was retained.

Since typical Web Services are stateless, the functionality was reduced to the minimum set of methods to exchange OPC Data Access information, without the need for methods to create and modify a context for communication.[27]

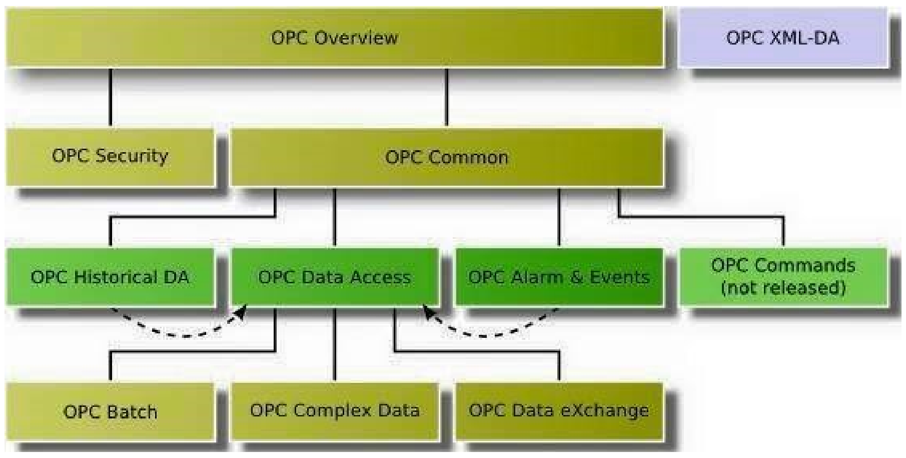


Figure 2.8: OPC standards tree [28]

2.3.3 OPC UA

OPC UA (the next generation OPC) is a new communication technology standard that was first released by the OPC Foundation in 2006 as an improvement on its predecessor, OPC Classic. OPC UA includes all the functionality found in OPC Classic. This is done by bringing together the different specifications of OPC Classic into a single entry point to a system offering current DA and A&E, combined with the history of both.

Furthermore, OPC UA is based on a cross-platform, business-optimized Service-Oriented Architecture (SOA), which expands on the security and functionality found in OPC Classic instead of the Microsoft-based COM/DCOM technology. OPC UA supports two protocols: a binary protocol that employs minimal resources, allowing for easy enablement through a firewall, and a web service protocol (SOAP) that uses standard HTTP/HTTPS ports.[29]

The most important benefits of using OPC UA:

- Expand security features.
- Limit configuration costs.
- Used for supervisory control.
- Easy to integrate into pre-existing IT networks.

Figure 2.9 describes the main differences between the classic OPC and next generation OPC.

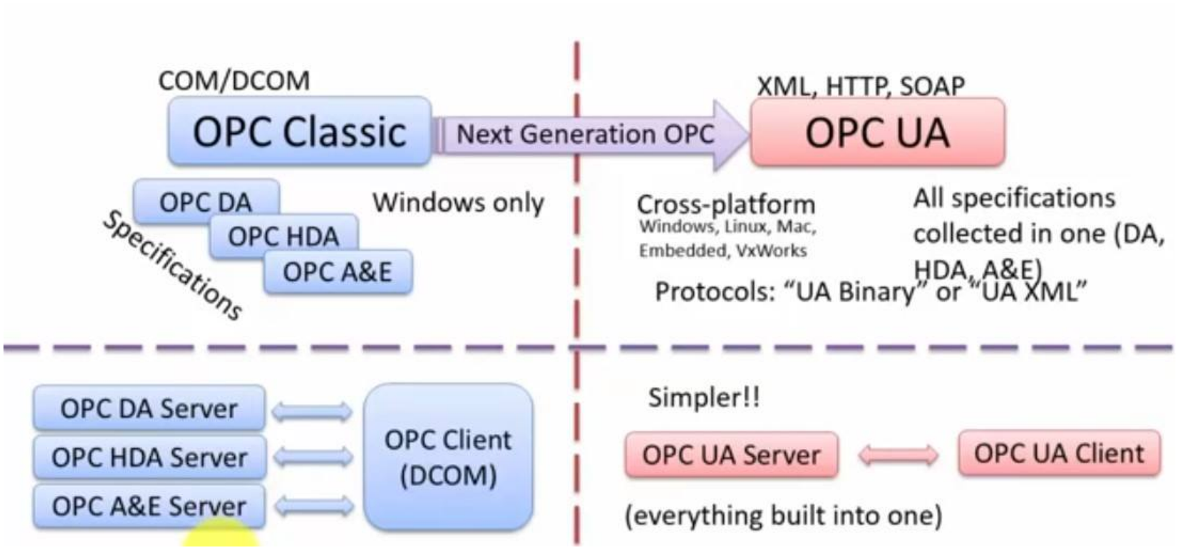


Figure 2.9: Differences between OPC classic and OPC UA

2.4 Modbus communication procotol

The Modbus protocol was created in 1979 by Modicon as a means of sharing data between their PLCs. Although initially a proprietary protocol controlled only by Modicon, it is since 2004 controlled by a community of users and suppliers of automation equipment, known as Modbus-IDA.This nonprofit organization oversees the evolution of the protocol and seeks to drive its adoption by continuing to openly distribute the protocol specifications and providing an infrastructure for device compatibility certification. [30]

2.4.1 Modbus Data Models

The organization of the memory areas on the server to which the client may write to or read from is defined by the Modbus protocol itself. There are four distinct memory areas; two of them are organized as 16-bit registers, whereas the other two are composed of arrays of single bits. Likewise, two of the memory areas provide read and write access permissions, while the other two may only be read from.

Memory Area Name	Address Range	Register Size (Bits)	Access Permission
Input Registers	1 – 65,536	16	Read
Holding registers	1 – 65,536	16	Read/write
Discrete inputs	1 – 65,536	1	Read
Coils	1 – 65,536	1	Read/write

Table 2.1: Memory Areas of the Modbus Data Model [30]

2.4.2 Modbus Protocol Architecture

The Modbus protocol is organized as a two-layer protocol (Figure 2.10).

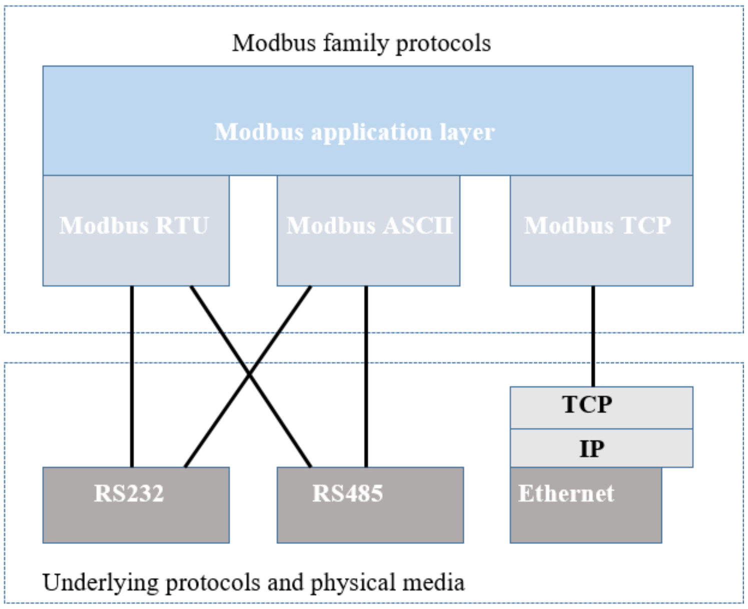


Figure 2.10: Organization of Modbus protocols [30]

The upper layer, called the “Modbus application layer”, defines the functions or services that a Modbus client may request of a Modbus server, and how these requests are encoded onto a message or APDU (application layer protocol data unit). It also defines how the servers have to reply to each function, and the actions they should take on behalf of the client.

The lower layer defines how the upper layer APDUs are encapsulated and encoded onto frames to be sent over the wire by the underlying physical layer, and how the server devices are addressed. There are three distinct versions of this lower layer. The ASCII (American Standard Code for Information Interchange) version encodes the upper layer APDU as ASCII characters, whereas the RTU and TCP versions use direct byte representation. The RTU and ASCII versions are expected to be sent over EIA/TIA-232 or EIA/TIA-485 (commonly known as RS232 and RS485). The TCP version, as the name implies, is sent over TCP connections established over the IP protocol. Although it is common that the IP protocol frames are then sent over an Ethernet LAN (local area network), there is no reason that they may not use any other underlying network, including the global Internet.

2.4.3 Modbus Application layer

The Modbus protocol defines three distinct APDUs (Figure 2.11) used in the Modbus application layer. All three APDUs start with a single-byte value indicating the Modbus function being requested or to which a reply is being made. Following the "function" byte value come all data and parameters of that specific function. Data and parameters have a variable number of bytes, depending on the function in question, and the number of memory registers that are being accessed. All APDUs are, however, limited in size to a maximum of 253 bytes, due to limitations imposed by the underlying EIA/TIA-485 layer. [30]

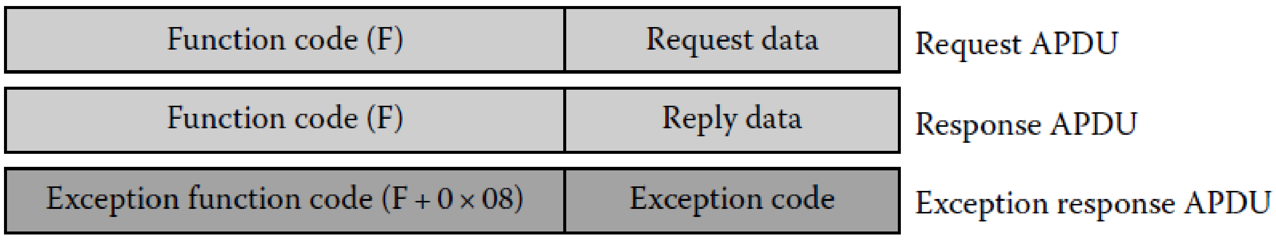


Figure 2.11: General format of APDU frames [30]

2.4.3.1 Data Access Functions

The Modbus protocol defines a large list of functions. The most often used functions are those associated with accessing the memory areas.

All functions listed in Table 2.2, with the exception of functions 0x14, 0x15, 0x16, and 0x18,simply request that some data be read or written to a specific memory area. Function codes 0x05 and 0x06 are used to write to a single element (coil or holding register, respectively). The remaining functions allow the client to read from or write to multiple contiguous elements of the same memory area.

Note that the maximum number of addressable elements is limited by the maximum size of the APDU.

Function 0x16 (Mask Write Register) changes the value stored in a single holding register. The new value is obtained by applying a logical operation using the current value of the holding register.

Function 0x18 (Read FIFO Queue) allows the client to request reading a FIFO queue from the server. The FIFO queue must be stored within the holding registers memory area in the server and consists of a first register containing the number of elements in the queue (queue count register), followed by the values of each element in the queue in the following registers (queue data registers).

The Modbus protocol includes two additional functions for data access. These functions (0x14 for reading and 0x15 for writing) are referred to in the base Modbus specification documents as “Read File Record” and “Write File Record”. Note that the memory referenced by these files is independent from the memory areas defined in Table 2.1.

Memory Area	**Function Name**	**Function Code(Hex)**	**Addressable Elements**	**Possible Response Error Codes**
Discrete Inputs	Read discrete inputs	0x02	1 – 2000	01, 02, 03, 04
Coils	Read coils	0x01	1 – 2000	01, 02, 03, 04
Coils	Write single coil	0x05	1	01, 02, 03, 04
Coils	Write multiple coils	0x0F	1 – 1976	01, 02, 03, 04
Input registers	Read input registers	0x04	1 – 125	01, 02, 03, 04
Holding registers	Read holding registers	0x03	1 – 125	01, 02, 03, 04
Holding registers	Write single register	0x06	1	01, 02, 03, 04
Holding registers	Write multiple registers	0x10	1 – 123	01, 02, 03, 04
Holding registers	Read/write multiple registers	0x17	1– 121 (write) 1 – 125 (read)	01, 02, 03, 04
Holding registers	Mask write register	0x16	1	01, 02, 03, 04
Holding registers	Read FIFO queue	0x18	1 – 32	01, 02, 03, 04
Files	Read file record	0x14		01, 02, 03, 04, 08
Files	Write file record	0x15		01, 02, 03, 04, 08

Table 2.2: Functions Used for Data Access [30]

2.4.3.2 Diagnostic Functions

The second large group of Modbus functions allows the client to obtain diagnostic information from the server (Table 2.3).

With function 0x07 (Read Exception Status) the client may read 8 bits of device-specific exception status information. Function 0x08 (Diagnostic) is used to obtain diagnostic information from the server.The function is followed by a subfunction code indicating which diagnostic information is being requested (bus message count, communication error count, etc.), or which diagnostic routine should be executed (restart communication, force listen only mode, etc.).

With function 0x0B (Get Communication Event Counter) the client can obtain a status word as well as an event count of the server's communication event counter. Function 0x0C (Get Communication Event Log) returns the same data as function 0x0B, plus the message count (number of messages processed since last restart) and a field of 64 event bytes.

With function 0x11 (Report Slave ID), a client may obtain the run status of the server device (run/ stop), a device-specific identification byte, and some additional device-specific 249 bytes of data.[30]

Function Name	**Function Code (Hex)**	**Possible Response Error Codes**
Read exception status	0x07	01, 04
Diagnostic	0x08	01, 03, 04
Get communication event counter	0x0B	01, 04
Get communication event log	0x0C	01, 04
Report slave ID	0x11	01, 04

Table 2.3: Modbus Function Codes for Diagnostic Purposes [30]

2.4.3.3 Error Handling

Error checking starts as soon as the server receives a request APDU. At this time it will start off by verifying the validity of the function code, address values, and data values (in this order) and, upon the first error encountered, will reply with an exception response APDU with error codes 1, 2, or 3, respectively (Table 2.4).

Code (Hex)	**Name**	**Comments**
0x01	Illegal function	
0x02	Illegal data address	
0x03	Illegal data value	
0x04	Slave device failure	
0x05	Acknowledge	Not commonly used. Indicates that the slave will reply later to the request.
0x06	Slave device busy	
0x08	Memory parity error	Used only in function codes 0x14 and 0x15.
0x0A	Gateway path unavailable	Only used by gateways.
0x0B	Gateway target device failed to respond	Only used by gateways.

Table 2.4: Modbus Exception Codes [30]

2.4.4 Modbus Serial

Modbus serial follows the master–slave interaction model, with the Modbus client becoming the master, and the Modbus servers taking the role of slaves. The master is responsible for initiating the communication by sending requests to the slaves, one request at a time. These, in turn, reply to the master with the requested data. The request/reply exchange can be performed in one of two ways:

- *Unicast mode:* The master sends a request to a specific slave. The slave processes this request and replies to the master.
- *Broadcast mode:* The master sends a request to all slaves. The slaves process this request, but do not reply to the master

The master only starts a request/reply exchange once the previous exchange has finished. Each Modbus serial network may only have one master. The number of slaves is limited to 247. In Modbus serial, frames exchanged between master and slave devices can be transmitted in one of two modes: RTU or ASCII, where RTU mode is the commonly used one in industry.

2.4.4.1 RTU mode frame

The RTU mode uses an asynchronous approach for data transmission. Each byte, within a frame, is transmitted using an 11 bit character (Figure 2.12):

- 1 start bit (ST), used for the initial synchronization.

- 8 bits, the data coded in binary with the least significant bit sent first.
- 1 parity bit (PT), used for error detection.
- 1 stop bit (SP), to ensure a minimum idle time between consecutive character transmissions.

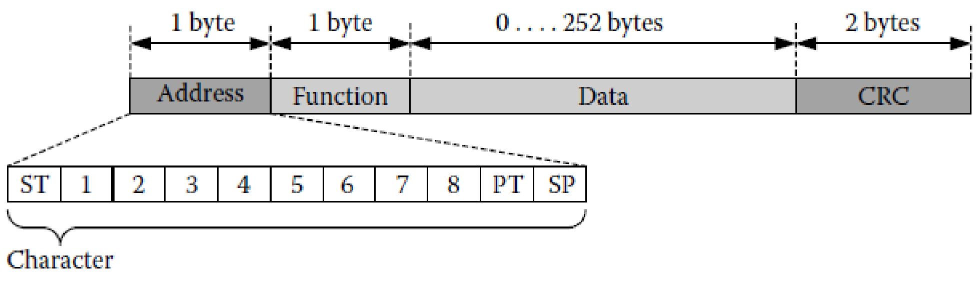


Figure 2.12: RTU frame transmission [30]

2.4.5 Modbus TCP

A Modbus TCP network consists of multiple devices connected through a TCP/IP network, interacting following the client–server model. A client sends a request to a server, which in turn responds to the client with the requested data. This transaction (request/response exchange) is performed by sending Modbus TCP frames through a TCP connection previously established between the client and the server. Connection establishment and management are handled by the TCP/IP protocol and occur independently of the Modbus protocol. A Modbus server listens on port 502 for requests from clients that wish to establish a new connection with the server. This port is presently reserved (and registered) for Modbus applications.

2.4.5.1 Modbus TCP frame

Modbus TCP frames (request and response) consist of a MBAP header (Modbus Application Protocol header) plus the application layer APDU (Figure 2.13).

The MBAP header comprises several fields:

- *Transaction identifier:* Since a client can issue several concurrent transactions over the same TCP connection, the responses are not guaranteed to arrive in the same order that the requests were sent. It is therefore necessary to have an identifier for each transaction in order to match the request and response frames. The client initializes this field when it performs a request. The server echoes this value in the response frame.

- *Protocol identifier:* This is used to identify the protocol; it currently always has the value 0.
- *Length:* This field indicates the size (in bytes) of the unit identifier field plus the APDU. Data transfer in a TCP connection is performed as a stream of bytes, which could lead to a situation where several frames are waiting to be read in the reception buffer. To identify frame boundaries in these situations, the frame length must be known.
- *Unit identifier:* This field is used to identify the destination device (a slave). It is used mainly by gateways between Modbus/TCP and Modbus serial networks, where the gateway, upon receiving a Modbus/TCP frame, needs to know the identification of the slave on the Modbus serial network that should receive that frame. Modbus/TCP devices that are not gateways usually ignore this field.

Unlike Modbus serial, Modbus TCP frames do not have an error detection field. This was considered unnecessary as the TCP/IP stack already includes several error detection mechanisms.

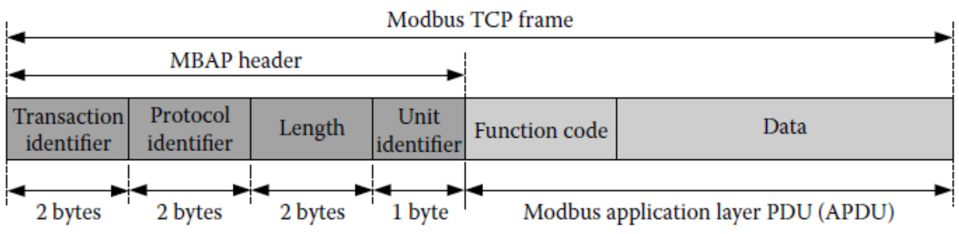


Figure 2.13: Modbus TCP frame [30]

Conclusion

Our contributions of this chapter were directed to achieve indirect connection between the MAS and hardware to develop CPS approach. First, we developed the idea of using agents to discrete event processes control. Second, we presented a concrete solution for application development using JADE environment. Then a practical solution for connecting the JADE platform indirectly with hardware using OPC standard, socket network and Modbus protocol.

Chapter 3

Hardware-In-the-Loop Simulation

Introduction

The verification and validation is a crucial step to fill the gap between conception and real-time operation, before execution of a software in the industrial environment.

Hardware-In-the-Loop Simulation (HILS) approach is considered as a promising validation of the software by providing a desired degree of reliability and flexibility. It is based on the integration of physical devices and electronic components (PLC, microcontroller, RFID technologies) in the 3D simulation loop.

In this study, HILS approach is used to validate the DES model by integrating all devices that aid the approach works properly as: RFID, PLC and Arduino.

This chapter will give an overview on HILS approach, introduce the different hardware elements integrated, and explain the interactions and the developed communication tools between them.

3.1 HILS validation technique

Hardware-In-the-Loop Simulation (HILS) is a type of real-time simulation. We use HILS simulation to test a controller design. It shows how a controller responds, in real time, to realistic virtual stimuli. We also use HILS to determine if our physical system (plant) model is valid.

In HILS technique, we use a real-time computer as a virtual representation of our plant model and a real version of our controller.
A typical HILS setup is shown in Figure 3.1.

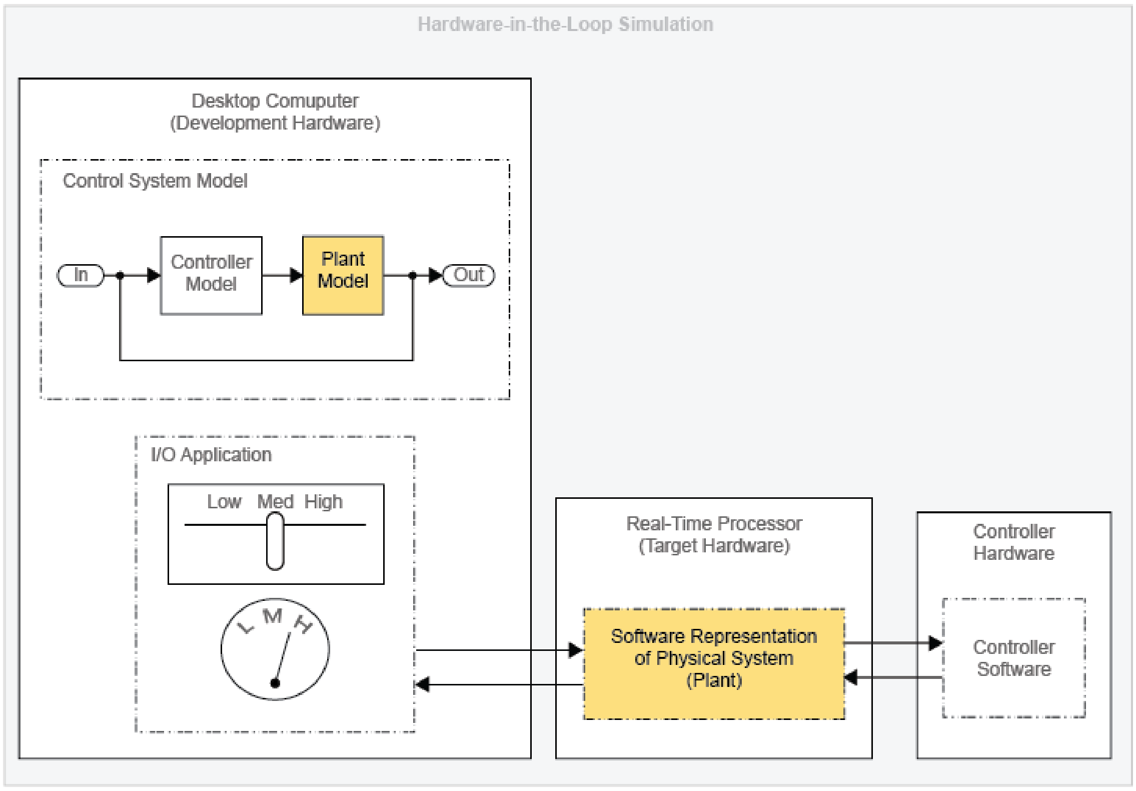

Hardware-in-the-Loop Simulation
Desktop Comuputer (Development Hardware)
Control System Model
In
Controller Model
Plant Model
Out
I/O Application
Low Med High
L M H
Real-Time Processor (Target Hardware)
Software Representation of Physical System (Plant)
Controller Hardware
Controller Software

Figure 3.1: A typical HILS setup [31]

The desktop computer (development hardware) contains the real-time capable model of the controller and plant. The development hardware also contains an interface with which to control the virtual input to the plant. The controller hardware contains the controller software that is generated from the controller model. The real-time processor (target hardware) contains code for the physical system that is generated from the plant model.[31]

3.1.1 HILS vs SILS

Another technique used for the same aim as HILS is the so called Software-In-the-Loop Simulation (SILS)

3.1.1.1 SILS validation technique

Software-In-the-Loop Simulation (SILS) is simply replacing the hardware equipments used previously in HILS approach (processor,controller and other equipments depending on the application) by a simulation.

The SILS technique is often used in the design phase of control architecture in order to adjust the control functionality and to evaluate its performance. This technique results some advantages [32]:

- Speed up designing, implementation and debugging stage.
- Reducing time and cost in the design stage.
- Ability to exercise a much greater portion of the possible scenario space than empirical testing.

3.1.2 HILS advantages

HILS technique can bring more fidelity to the control validation by maintaining the behavior of the real system. This technique consists in integrating hardware equipment in the closed loop with a simulation model. Two reasons argue the use of this technique. The first one, a direct test on real manufacturing system can be not suitable because of high safety risks and/or expensive test time and/or destructive test (products, equipment). The second one, the simulation is not sufficiently accurate to mimic perfectly manufacturing system, which leads to a lack of confidence in the validation of the control system. In addition to the advantages inherited from SILS approach, some other advantages of this technique are summarized as follows:

- Verification and validation of the complex control software in the presence of more realistic conditions.
- Practice in a quasi-real industrial environment at the scale of a laboratory.

In some systems, both SILS and respectively HILS are used in the design validation respecting the workflow shown in Figure 3.2 .

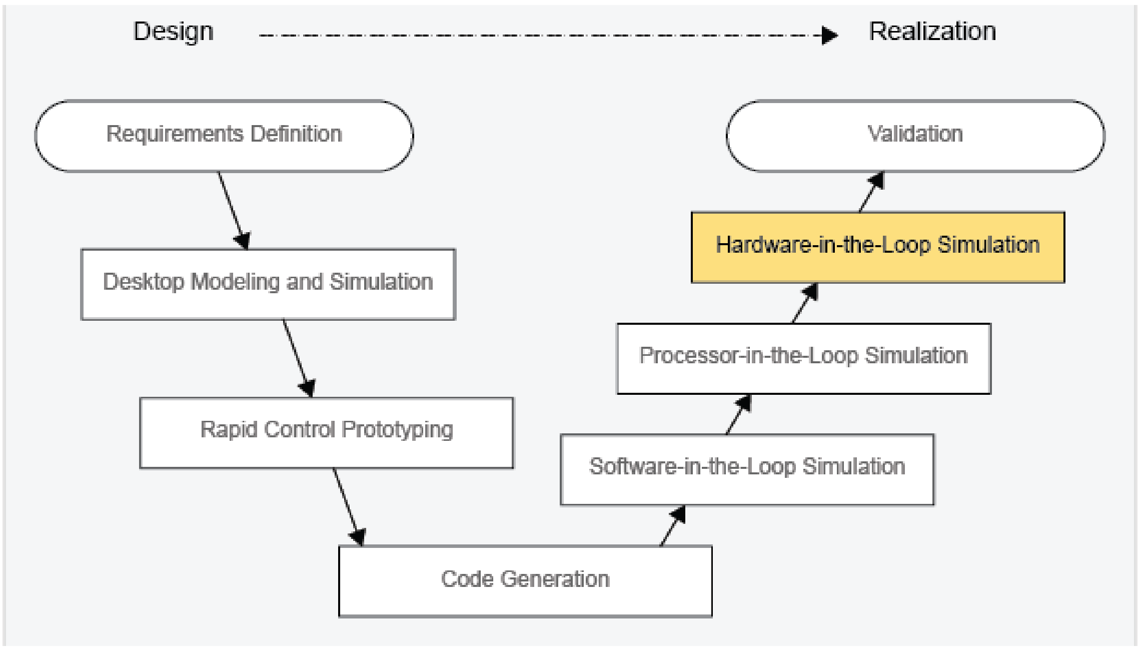

Design
Realization
Requirements Definition
Desktop Modeling and Simulation
Rapid Control Prototyping
Code Generation
Software-in-the-Loop Simulation
Processor-in-the-Loop Simulation
Hardware-in-the-Loop Simulation
Validation

Figure 3.2: System design till realization workflow [31]

3.2 The Programmable logic controller (PLC)

A programmable logic controller (PLC) is an industrial grade computer that is capable of being programmed to perform control functions. The programmable controller has eliminated much of the hardwiring associated with conventional relay control circuits. Other benefits include easy programming and installation, high control speed, network compatibility, troubleshooting and testing convenience, and high reliability.[33]

(a) Siemens s7-1200 compact PLC

(b) Siemens s7-300 modular PLC

Figure 3.3: Siemens s7-1200 and s7-300 PLCs

A PLC exists in two distinguishable forms: compact and modular.

- **A compact PLC**: where all its elements are integrated like inputs/outputs, power supply, communication unit...(Figure 3.3a) . This type of PLCs is made usually for small automatic systems.
- **A modular PLC**: where its elements are seperated units (Figure 3.3b). This type of PLCs is used in powerful and complex systems.

3.2.1 PLC architecture and elements

Figure 3.4 shows the basic internal architecture of a PLC. It consists of a central processing unit (CPU) containing the system microprocessor, memory, and input/output circuitry.[34]

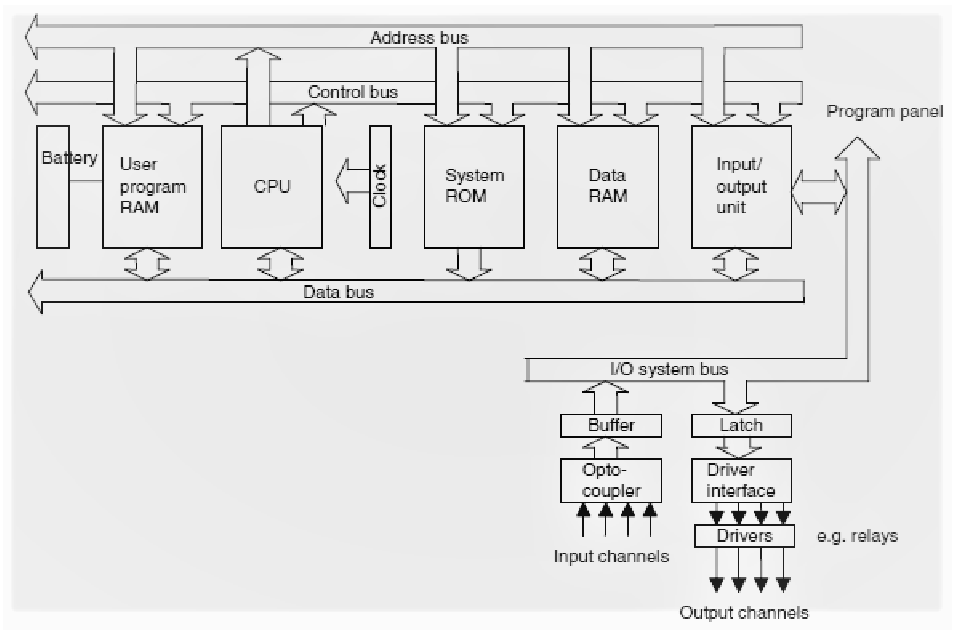


Figure 3.4: PLC bacic architecture [34]

3.2.1.1 The Central processing unit (CPU)

It is the brain of a PLC, its internal structure comprises:

- Arithmetic and logic unit (ALU) that is responsible for data manipulation and carrying out arithmetic operations of addition and subtraction and logic operations of AND, OR, NOT, and EXCLUSIVE-OR.
- Memory, termed registers, located within the microprocessor and used to store information involved in program execution.
- A control unit that is used to control the timing of operations.

3.2.1.2 Buses

Buses are the paths used for communication within the PLC. The information is transmitted in binary form, that is, as a group of bits. The system has four buses:

- The data bus carries the data used in the processing done by the CPU
- The address bus is used to carry the addresses of memory locations.
- The control bus carries the signals used by the CPU for control, such as to inform memory devices whether they are to receive data from an input or output data and to carry timing signals used to synchronize actions.
- The system bus is used for communications between the input/output ports and the input/output unit.

3.2.1.3 Memory

To operate the PLC system there is a need for it to access the data to be processed and instructions, that is, the program, which informs it how the data is to be processed. Both are stored in the PLC memory for access during processing. There are several memory elements in a PLC system:

- The data bus carries the data used in the processing done by the CPU
- System Read-only memory (ROM) gives permanent storage for the operating system and fixed data used by the CPU.
- A Random-access memory (RAM) is used for the user's program.
- A Random-access memory (RAM) is used for data
- Possibly, as an extra module, Erasable and programmable read-only-memory (EPROM) is used to store programs permanently.

3.2.1.4 Input/Output Unit

The input/output unit provides the interface between the system and the outside world, allowing for connections to be made through input/output channels to input devices such as sensors and output devices such as motors and solenoids. It is also through the input/output unit that programs are entered from a program panel.A small PLC is likely to have just one form of input/output, such as 24 v. Outputs are specified as being of relay type, transistor type(fastest switching action), or triac type.

3.2.2 PLC programming

Each input and output PLC module terminal is identified by a unique address. In PLCs, the internal symbol for any input is a contact. Similarly, in most cases, the internal PLC symbol for all outputs is a coil. The standard IEC 61131 was established to standardize the multiple languages associated with PLC programming (Figure 3.5) by defining the following five standard languages:

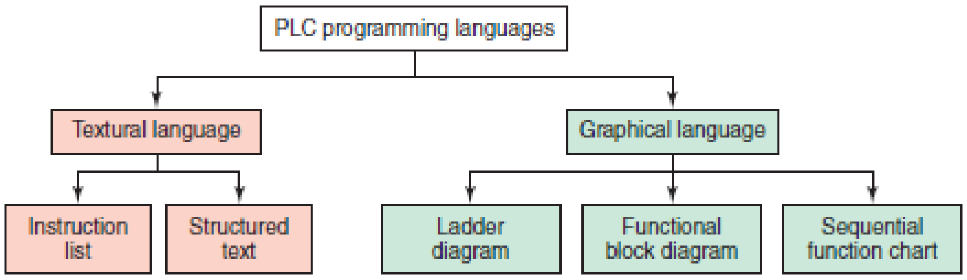


Figure 3.5: Standard IEC 61131 languages associated with PLC programming [33]

- **Ladder Diagram (LD)** a graphical depiction of a process with rungs of logic, similar to the relay ladder logic schemes that were replaced by PLCs.
- **Function Block Diagram (FBD)** a graphical depiction of process fl ow using simple and complex interconnecting blocks.
- **Sequential Function Chart (SFC)** a graphical depiction of interconnecting steps, actions, and transitions.
- **Instruction List (IL)** a low-level,text-based language that uses mnemonic instructions.
- **Structured Text (ST)** a high-level,text-based language such as BASIC, C, or PASCAL specifically developed for industrial control applications.

3.2.3 Siemens ET 200SP PLC

ET 200SP PLC (Figure 3.6) is one of the recent releases of Siemens (04/2012). It is mainly designed for complex distributed input/output systems to be employed as a slave to a master controller taking advantage of its high scalability (possibility to be extended in a modular form)(Figure 3.6b) and flexibility. It can be operated as a standalone controller too (Figure 3.6a).

(a) ET 200SP PLC

(b) Scaled ET 200SP PLC

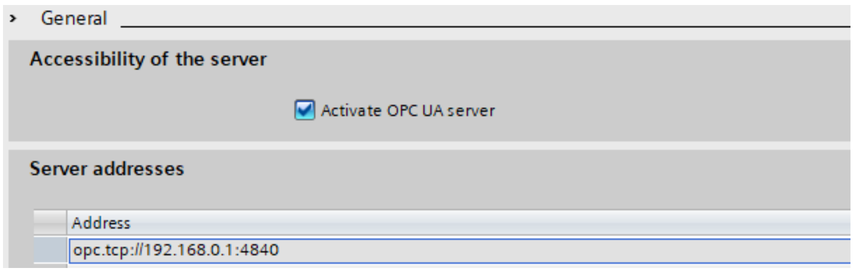
Figure 3.6: Siemens ET 200SP PLC

Table 3.1 summerizes some important specifications about this PLC given by the manufacturer.

CPU processing time	
bit operation word operation fixed-point arithmetic floating-point arithmetic	48 ns 58 ns 77 ns 307 ns
Supply voltage	
type of supply voltage low limit of permitted range(DC) high limit of permitted range(DC) reverse polarity protection	24V(DC) 19.2V 28.8 V Yes
Input current	
current consumption (rated value) Inrush curent (maximum)	0.6A 4.7A
Interfaces	
number of Profinet ports number of input/output	3 (1 integrated 2 via module) 0 integrated, up to 64 i/o module
Software engineering	
Software IDE Programming languages supported	TIA Portal v13 LAD,SFC,FBD,ST,IL

Table 3.1: Some technical specifications of ET200SP PLC [35]

The major advantage to use this PLC in this study is the ease of OPC UA server creation (Figure 3.7), hence PLC tags can be mapped easily to the emulation model in Flexsim to drive the virtual FMS.


General

Accessibility of the server

☑ Activate OPC UA server

Server addresses

Address
opc.tcp://192.168.0.1:4840

Figure 3.7: OPC UA server activation on ET 200SP PLC

3.3 Arduino Uno microcontroller board

Microcontroller (μC) is a term used to describe a system that includes a minimum of microprocessor, program memory, data memory and input-output (I/O). Some microcontroller systems also include timers, counters, analog to digital (A/D) converters and so on.[36]

Arduino Uno board (Figure 3.8) holds the 8-bit ATmega328 μC based on AVR architecture. It contains everything needed to support the microcontroller: 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs (A0-A5) provide Analog Digital Converter (ADC) with 10bits resolution, a 16 MHz crystal oscillator, a USB connection, a power jack, an ICSP header, and a reset button.

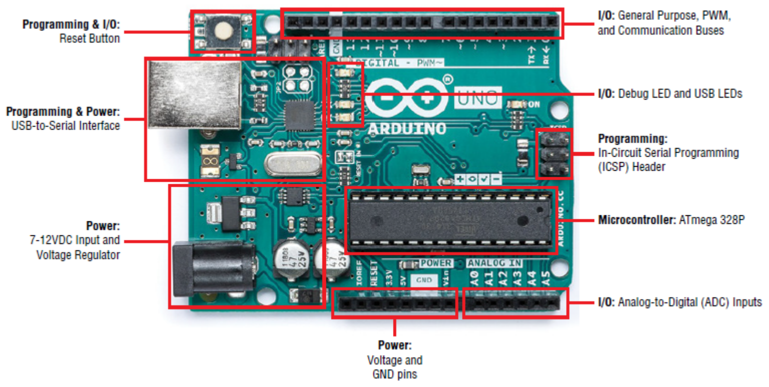

Programming & I/O: Reset Button
Programming & Power: USB-to-Serial Interface
Power: 7-12VDC Input and Voltage Regulator
Power: Voltage and GND pins
I/O: General Purpose, PWM, and Communication Buses
I/O: Debug LED and USB LEDs
Programming: In-Circuit Serial Programming (ICSP) Header
Microcontroller: ATmega 328P
I/O: Analog-to-Digital (ADC) Inputs

Figure 3.8: Arduino Uno board composition

The Arduino Uno board specifications are summarized in table 3.2

Specification	Parameter
Microcontroller	ATmega328
Operating voltage	5V
Input voltage	7-12 V
Digital I/O pins	14
Analog input pins	6
DC current per input pin	40 mA
DC current for 3.3V pin	50 mA
Flash memory	32 Kb
SRAM	2 Kb
EEPROM	1 Kb
Clock speed	16kHz

Table 3.2: Arduino uno specifications [37]

3.3.1 Ethernet Shield W5100

The Arduino Ethernet Shield W5100 (Figure 3.9) allows an Arduino board to connect to the internet. It is based on the Wiznet W5100 ethernet chip. The Wiznet W5100 provides a network (IP) stack capable of both TCP and UDP. It supports up to four simultaneous socket connections.[38]

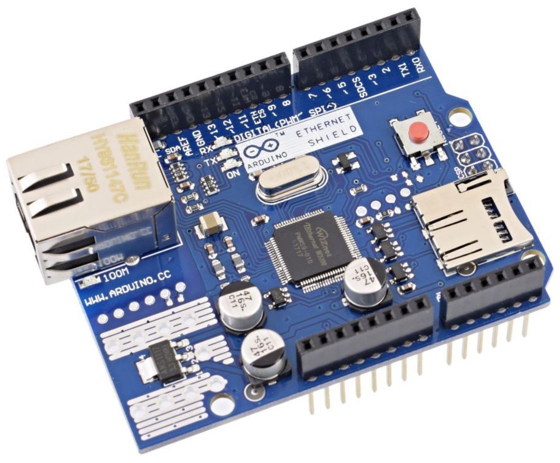


Figure 3.9: Ethernet shield W5100

The Ethernet shield connects to an Arduino board using long wire-wrap headers which extend through the shield (Figure 3.10). This keeps the pin layout intact and allows another shield to be stacked on top. Arduino Uno communicates with the shield using the SPI bus. This is on digital pins 11, 12, and 13. Pin 10 is used as SS pin.

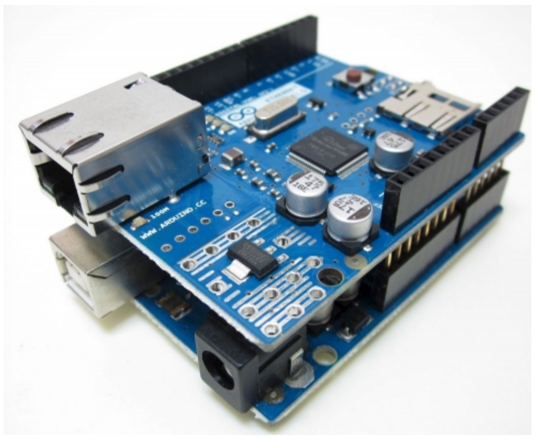


Figure 3.10: Ethernet shield W5100 plugged in an Arduino Uno board

3.3.2 ”Ethernet.h” Library

The open source Arduino library dealing with Ethernet communication is ”*Ethernet.h*.” To initialize the Ethernet library and network settings the function ”*Ethernet.begin()*” that receives as arguments MAC, IP, Gateway and Subnet addresses.[39]

3.4 Communication between Arduino Uno and ET 200SP PLC using Modbus TCP protocol

As described in the previous chapter, Modbus TCP communication protocol follows a client-server model. In our project, ET 200SP PLC is the client and Arduino Uno is the server. Hence, Arduino listens on port 502, wait for the PLC to connect and then start data exchange according to a Modbus function.

3.4.1 Server (Arduino Uno) part

In addition to ”*Ethernet.h*”, Arduino offers ”*ModbusIP.h*” library to deal with Modbus TCP communication.

3.4.1.1 ”ModbusIP.h” Library

This library implements Modbus protocol over TCP transport type. The main functions are described Table 3.3.

Function	**Description**
config(MAC,IP)	Receives MAC and IP adresses as argument to configure the Modbus TCP network.
task()	Maintain the Modbus TCP communication. It is called once in *Void loop()*.
addCoil(coilPIN,STATE)	Receives the coil output pin number and a boolean value to be read/written.
addIreg(IREG,Value)	Receives the input register number and value as arguments.
addHreg (HREG, Value)	Receives the holding register number and value to be read-/written as arguments.

Table 3.3: Main useful functions ”ModbusIP.h” library

In our project, we send and receive integer numbers, consequently, we select holding registers (see Table 2.1). Arduino reads one Holding Register (to which the PLC writes) and writes to another one (which the PLC reads).

A proposed Arduino sketch to perform this task is as follows:

```
#include <Ethernet.h>
#include <ModbusIP.h>

const int WRITE_HREG=100;//The number of the holding
                          //register where we write
const int READ_HREG=101;//The number of the holding
                         //register to be read
ModbusIP mb; //modbus object declaration
byte mac[] = {0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED};
byte ip[]={192, 168, 0, 5};
byte gateway[]={192, 168, 0, 200};
byte subnet[]={255, 255, 255, 0};

void setup()
{
   Ethernet.begin(mac,ip,gateway,subnet);
   mb.config(mac,ip);//configuration of the  Modbus TCP
                     //network
   mb.addHreg(WRITE_HREG,127);//Holding  registers
                              //configuration.
```

```
                                        //The initial value can be
                                        //any number.
    mb.addHreg(READ_HREG,127);
}

void loop()
{
    mb.task();//maintain Modbus communication and
                    //data update
}
```

Listing 3.1: Arduino sketch for the desired Modbus TCP communcation

3.4.2 Client (ET 200SP PLC) Part

In TIA portal (see appendix B), each Modbus TCP session can handle only one function. In our case, two Modbus functions are executed: read a holding register and write to a holding register; and hence two communication sessions are needed. Using Ladder logic diagram, a Modbus TCP client session is a single function block ”MB_CLIENT”. Hence to achieve our communication objective, two ”MB_CLIENT” blocks are required: one with function read a holding register (Figure 3.11), and the other with function write to a holding register (Figure 3.12).

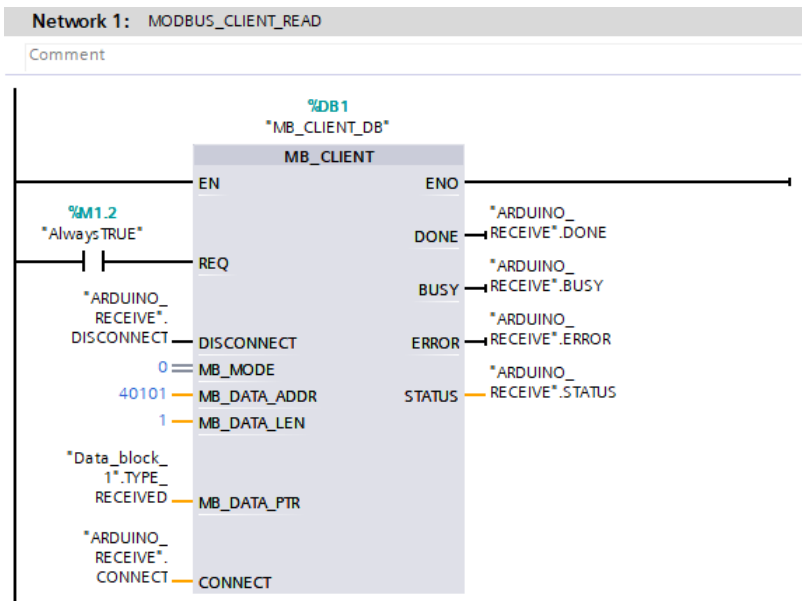


Figure 3.11: Modbus TCP client read a holding register function block

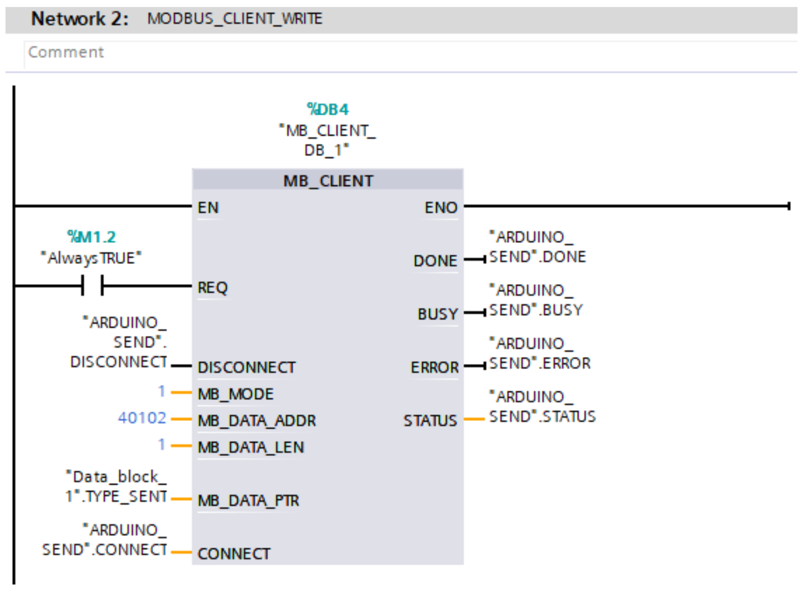


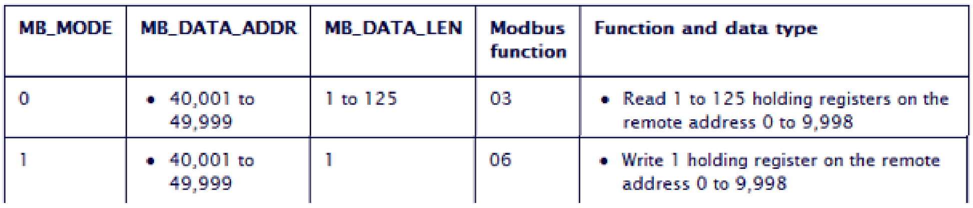
Figure 3.12: Modbus TCP client write to a holding register function block

3.4.2.1 Description and specifications of ”MB_ CLIENT” function block

”MB_ CLIENT” function block has a set of parameters:

- REQ (Boolean): Modbus query to the Modbus TCP server.As long as the input is set (REQ=true), the instruction sends communication requests, that’s why we attached to its input a memory zone that is always 1 (Always TRUE).
- Disconnect (Boolean): controls the establishment and termination of the connection to the Modbus server: 0 establish connection, 1 disconnect.
- DONE (Boolean): set once Modbus job is completed without errors.
- BUSY (Boolean): 0 if no Modbus request in progress,1 if a Modbus request is being processed.
- ERROR (Boolean): 0 if no error occured.
- STATUS (Word):Detailed status information of the instruction.
- MB_DATA_PTR (Variant): Pointer to a data buffer for the data to be received from the Modbus server or to be sent to the Modbus server.
- CONNECT (Variant): Pointer to the structure of the connection description. The structure contains all information about the communication session: address of the server (Arduino IP address in our case), port (502) communication type (TCP/IP), physical interface ethernet port number...

The rest of parameters (MB_MODE, MB_DATA_ADDR, MB_DATA_LEN) depends on the Modbus function we want to execute, the functions needed in our project are shown in the following screen shot taken from TIA portal information system (Figure 3.13).



MB_MODE	MB_DATA_ADDR	MB_DATA_LEN	Modbus function	Function and data type
0	• 40,001 to 49,999	1 to 125	03	• Read 1 to 125 holding registers on the remote address 0 to 9,998
1	• 40,001 to 49,999	1	06	• Write 1 holding register on the remote address 0 to 9,998

Figure 3.13: Modbus function parameters from TIA Portal information system

In the first block (Figure 3.11), we set MB_MODE = 0 inorder to perform a read holding register function , MB_DATA_LEN = 1 because we need to read a single holding register. The starting address is set to 40101 since the register number selected in the server (Arduino) was 100, and since the memory zone for this function starts from 40001 then the selected register address is 40001 100 40101.

In the second block (Figure 3.12), we set MB_MODE = 1 inorder to perform a write to holding register function, MB_DATA _LEN = 1 because this function can write to only a single holding register.The starting address is set to 40101 since the register number selected in the server (Arduino) was 101, and since the memory zone for this function starts from 40001 then the selected register address is 40001 101 40102.

3.5 Radio frequency identification (RFID) technology

Radio frequency identification (RFID) is a wireless communication technology that is used to uniquely identify tagged objects. There are three basic components to an RFID system: (Figure: 3.14)

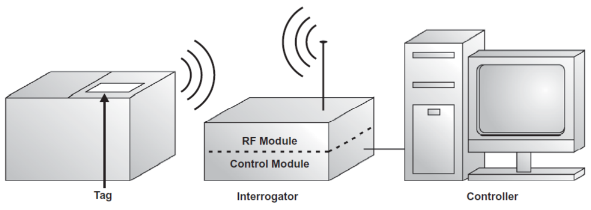


Figure 3.14: RFID basic building block

1. A tag (sometimes called a transponder), which is composed of a semiconductor chip, an antenna, and sometimes a battery.
2. An interrogator (sometimes called a reader or a read/write device),which is composed of an antenna, an RF electronics module, and a control electronics module.

3. A controller (sometimes called a host),which most often takes the form of a PC or a workstation running database and control (often called middleware) software.[40]

3.5.1 RFID tags

The basic function of an RFID tag is to store data and transmit data to the interrogator, and it can be classified according to its composition as:

3.5.1.1 Active vs. Passive Tags

RFID tags are said to be active if they contain an on-board power source, such as a battery.
Passive RFID tags have no on-board power source and derive power.

3.5.1.2 Read-only vs. read/write ”smart” tags

Having only a read-only memory, Read-only (RO) tags are similar to bar codes. They are programmed once, by a product manufacturer for instance, and from thereon cannot be altered.

Read/Write tags are often called “smart” tags. Smart tags present the user with much more flexibility than RO tags. They can store large amounts of data and have an addressable memory that is easily accessed and changed.[40]

3.5.2 RFID Interrogator

An RFID interrogator acts as a bridge between the RFID tag and the controller and has just a few basic functions:

- Read the data contents of an RFID tag
- Write data to the tag (in the case of smart tags)
- Relay data to and from the controller
- Power the tag (in the case of passive tags)

3.5.3 RFID Controller

RFID controllers are the “brains” of any RFID system. They are used to network multiple RFID interrogators together and to centrally process information.

3.5.4 Frequency of operation

A key consideration for RFID is the frequency of operation. RFID systems can use different bands for communication. (Figure 3.15)

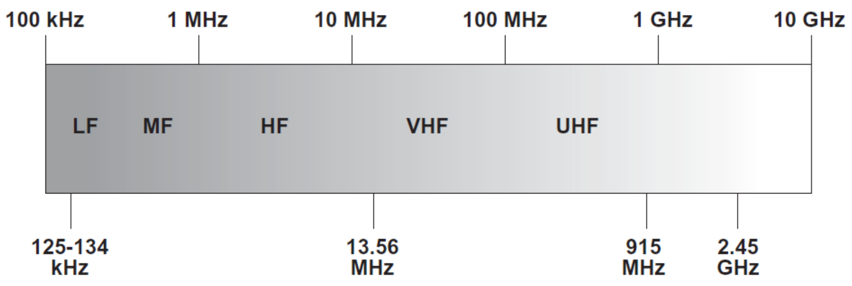


Figure 3.15: RF spectrum [40]

In RFID there are both low frequency and high radio frequency bands in use, as shown in the following list:

- **Low Frequency RFID Bands**
  - Low frequency (LF): 125–134 KHz
  - High frequency (HF): 13.56 MHZ
- **High Frequency RFID Bands**
  - Ultra-high frequency (UHF): 860–960 MHZ
  - Microwave: 2.5 GHz and above

The choice of frequency has a major effect on several characteristics of any RFID system such as: read range, the type of tags to be chosen ...[40]

3.5.5 RC 522 RFID module

The RC522 (Figure 3.16) is a 13.56MHz RFID module that is based on the MFRC522 controller from NXP semiconductors. The module can supports I2C, SPI and UART communications and normally is shipped with a RFID card and key fob. [41]

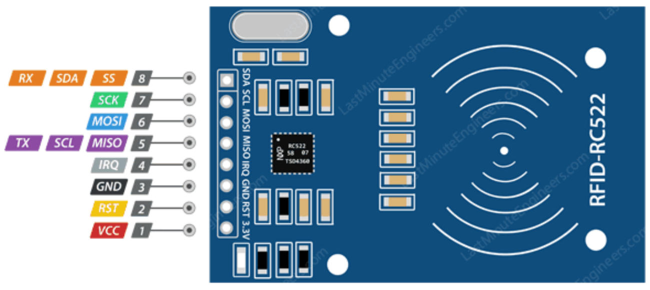


Figure 3.16: RC522 RFID module pinout

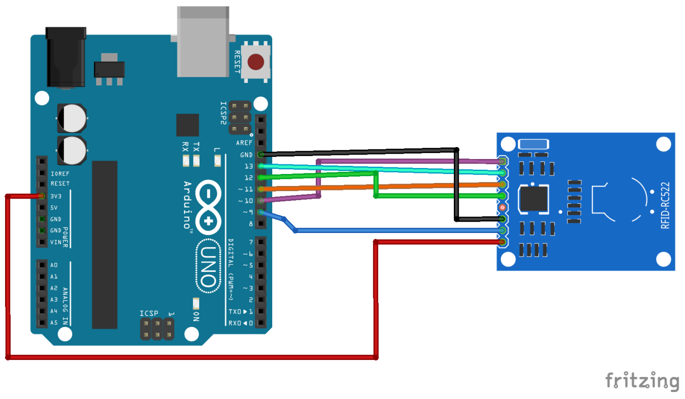
The pin specification of this RFID module is detailed in Table 3.4 .

Pin Number	Pin Name	Description
1	Vcc	Used to Power the module, typically 3.3V is used
2	RST	Reset pin – used to reset or power down the module
3	Ground	Connected to Ground of system
4	IRQ	Interrupt pin – used to wake up the module when a device comes into range
5	MISO/SCL/Tx	MISO pin when used for SPI communication, acts as SCL for I2c and Tx for UART.
6	MOSI	Master out slave in pin for SPI communication
7	SCK	Serial Clock pin – used to provide clock source
8	SS/SDA/Rx	Acts as serial input (Slave Select) for SPI communication, SDA for I2C and Rx during UART

Table 3.4: RC522 pin configuration [41]

3.5.5.1 Arduino Uno - RC522 interface

Figure 3.17 shows the connection between Arduino Uno and RC522 module.



Pin	Wiring to Arduino Uno
SDA	Digital 10
SCK	Digital 13
MOSI	Digital 11
MISO	Digital 12
IRQ	unconnected
GND	GND
RST	Digital 9
3.3V	3.3V

Figure 3.17: Arduino Uno-RC522 pins connection

In this project, RC522 RFID module uses the Serial Peripheral Interface (SPI) protocol to communicate with Arduino. SPI is a synchronous serial data protocol used by microcontrollers for communicating with one or more peripheral devices quickly over short distances. With an SPI connection there is always one master device (usually a microcontroller) which controls the peripheral device through four lines:

- MISO (Master In Slave Out) - The Slave line for sending data to the master.
- MOSI (Master Out Slave In) - The Master line for sending data to the peripherals.
- SCK (Serial Clock) - The clock pulses which synchronize data transmission generated by the master.
- SS (Slave Select) - the pin on each device that the master can use to enable and disable specific devices.[42]

3.5.5.2 MFRC522.h and SPI.h libraries

Inorder to program the interface and communication between Arduino and RC522 and perform an RFID read/write operation on a *Mifare* RFID tag, open source Arduino libraries MFRC522.h and SPI.h are offered. To initiate an SPI communication we use the function ”SPI.begin()”. The boolean function ”PICC_ReadCardSerial()” is used to detect whether a tag is within the RC522 range.

In our project, RFID tags are distinguished through their UIDs. The following Arduino scketch is proposed to read tags' UIDs.

```
#include <SPI.h>
#include <MFRC522.h>
#define SS_PIN 10
#define RST_PIN 9

MFRC522 mfrc522(SS_PIN, RST_PIN);//Create MFRC522 instance

void setup()
{
    Serial.begin(9600);// Initiate a serial communication
   SPI.begin();           // Initiate  SPI bus
   mfrc522.PCD_Init();    // Initiate MFRC522
}

void loop()
{
  if (!mfrc522.PICC_IsNewCardPresent())//Look for new cards
    return;
  if (!mfrc522.PICC_ReadCardSerial())//Select one of the
                                     //cards
    return;
  Serial.print("UID tag :");//Show UID on serial monitor
  String content= "";
  byte letter;
  for (byte i = 0; i < mfrc522.uid.size; i++)
  { //Print the UID in both decimal and hexadecimal format
     Serial.print(mfrc522.uid.uidByte[i]<0x10 ? " 0":" ");
     Serial.print(mfrc522.uid.uidByte[i], HEX);
  }
}
```

Listing 3.2: Arduino sketch for reading RFID tags' UIDs

Conclusion

In this chapter the composition and the interactions in the HILS system are clarified. We got firstly a general idea about HILS technique. Then, we introduced the hardware components integrated (ET 200SP PLC, Arduino Uno μC board with Ethernet Shield and RC522 RFID module). Finally, we described communication tools (Modbus TCP and OPC UA server activation) and interactions (RC522 connection and programming with Arduino) to link between those components and the virtual model. Our HILS system is now ready.

Chapter 4

Cyber-physical Product-driven System Implementation

Introduction

In the last decade of twenty century, the cyber world and the physical world were considered as two different entities. However, we can easily find that these two entities are closely correlated with each other after integrating sensors and actuators in the cyber systems.

Cyber systems became responsive to the physical world by enabling real time control issued from conventional computer systems, thus giving birth to a new research paradigm named Cyber-Physical System CPS that is considered as a building block of new industrial systems (Industry 4.0).

This chapter proposes a validation and practical framework of PDS applied to the highly Automated Flexible Robotized Assembly System (AIP-PRIMECA) relying on MAS as implementation framework, RFID as auto-identity technologies to provide the intelligent product concept (PDS) and HILS as practical real time monitoring of production lines as well as products. In addition, FMS scheduling to enhance the flexibility and reliability of AIP-PRIMECA.

4.1 Case study: AIP-PRIMECA FMS

AIP-PRIMECA [1] are resource centers which bring together the technical and human resources used as training support in the fields of Integrated Design in Mechanics and Production. There are ten regional AIP-PRIMECA poles in France.

Our MAS architecture of control is applied to an emulated FMS of AIP-PRIMECA located at Valenciennes site. This experimental support (Figure 4.1) is an academic technological platform which is considered as a real production cell.

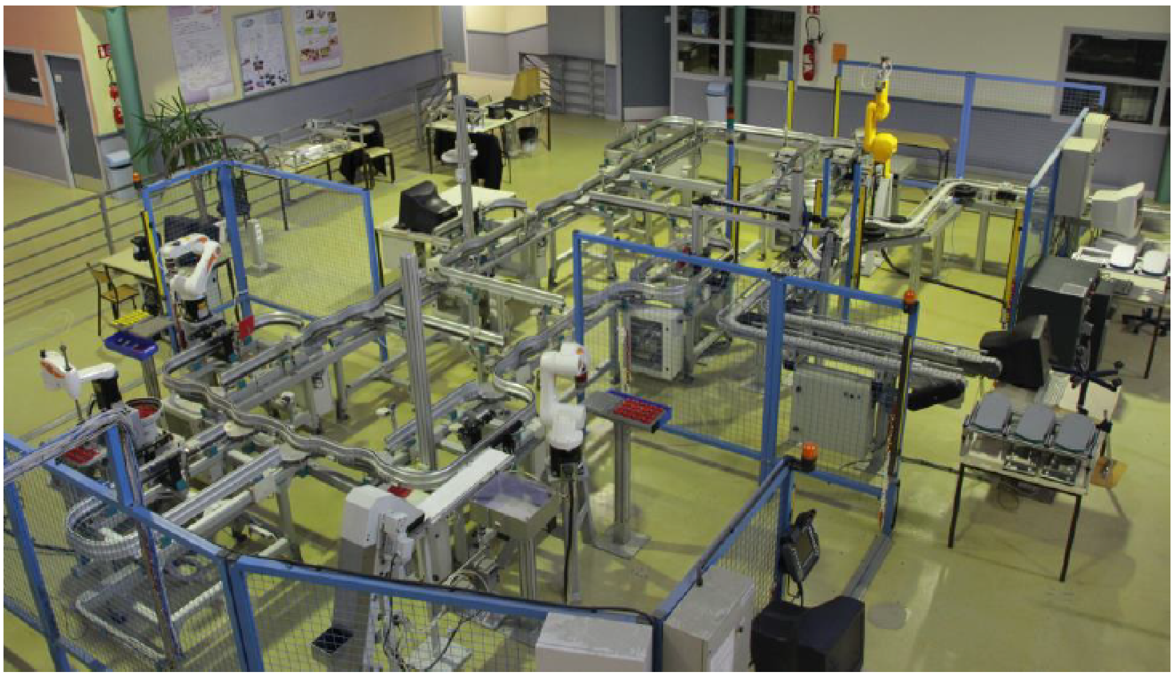


Figure 4.1: The real AIP-PRIMECA FMS of the case study

4.1.1 FSM data presentation

AIP-PRIMECA FMS is a fully robotically fexible assembly cell which is composed of six workstations: load/unload robotic workstation ($R1$); three assembly Kuka robots ($R2$, $R3$ and $R4$); a Cognex camera as an automated inspection workstation($M5$) and finally,two optional robotic workstations ($R6$ and $R7$) which are not used in this study.

[1]A french abbreviation whose translation is: Inter-establishment production and IT resource center for mechanics

Figure 4.2 is a presentation diagram of the studied FMS.

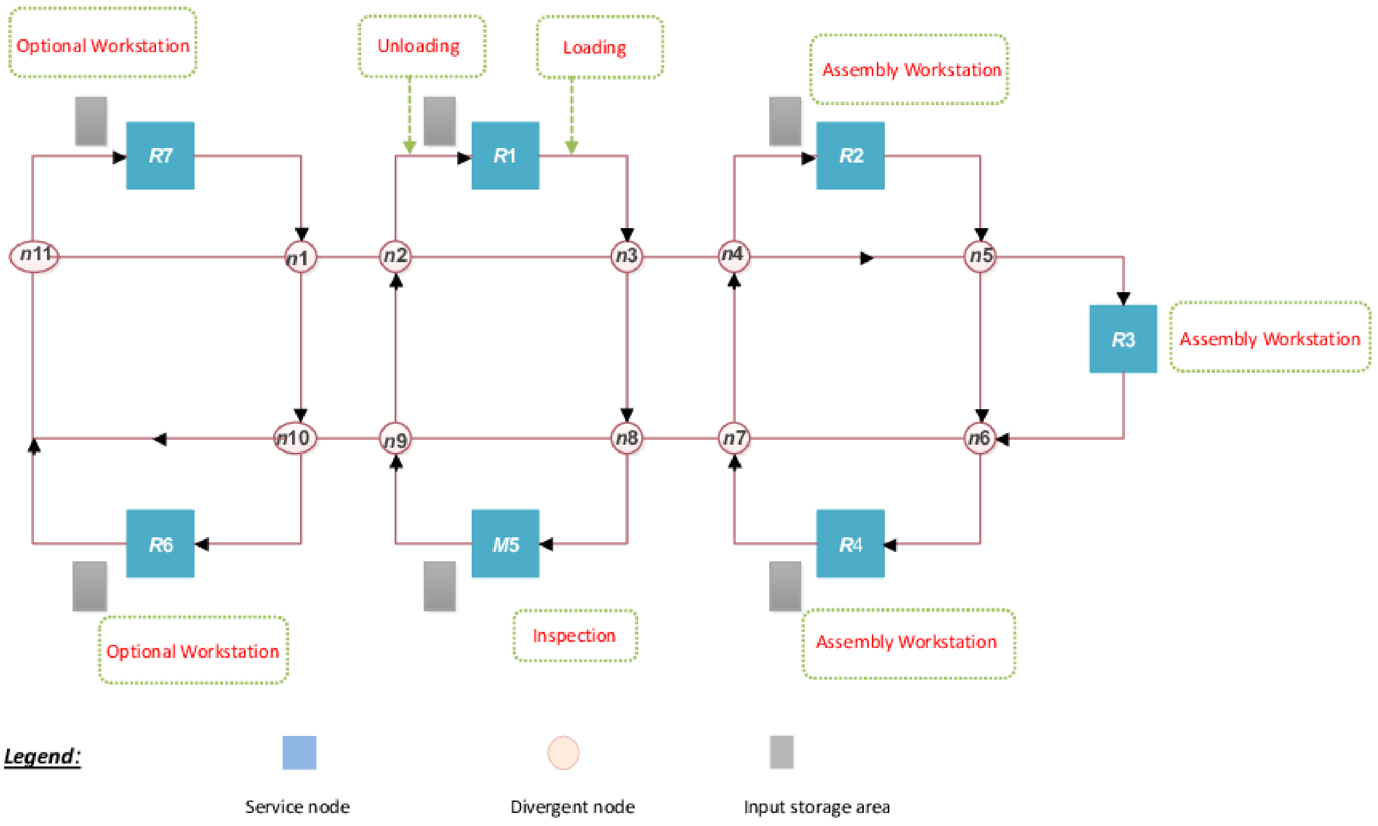


Figure 4.2: AIP-PRIMECA FMS presentation diagram

Shuttles are used to transport products between different workstations through unidirectional conveyor. Several routing nodes (divergence points) are indentified throughout the conveying system ($n1$, $n2$, $n3$, $n4$, $n5$, $n6$, $n7$, $n8$, $n9$, $n10$ and $n11$). These routing nodes are categorized to two types:machine routing nodes which are localized in front of the entrance of machines ($n2$, $n4$, $n5$, $n6$, $n8$, $n10$ and $n11$) and ordinary rooting nodes ($n1$, $n3$, $n7$ and $n9$).

Table 4.1 summarizes the transport time necessary for a shuttle to move between the different nodes of the cell.

	$n1$	$n2$	$n3$	$n4$	$n5$	$n6$	$n7$	$n8$	$n9$	$n10$	$n11$	$R1$	$R2$	$R3$	$R4$	$M5$	$R6$	$R7$
$n1$	▨	4	-	-	-	-	-	-		5	-	-	-	-	-	-	-	-
$n2$	-	▨	4	-	-	-	-	-	-	-	-	5	-	-	-	-	-	-
$n3$	-	-	▨	4	-	-	-	5	-	-	-	-	-	-	-	-	-	-
$n4$	-	-	-	▨	4	-	-	-	-	-	-	-	5	-	-	-	-	-
$n5$	-	-	-	-	▨	3	-	-	-	-	-	-	-	11	-	-	-	-
$n6$	-	-	-	-	-	▨	4	-	-	-	-	-	-	-	5	-	-	-
$n7$	-	-	-	5	-	-	▨	4	-	-	-	-	-	-	-	-	-	-
$n8$	-	-	-	-	-	-	-	▨	4	-	-	-	-	-	-	5	-	-
$n9$	-	5	-	-	-	-	-	-	▨	4	-	-	-	-	-	-	-	-
$n10$	-	-	-	-	-	-	-	-	-	▨	4	-	-	-	-	-	7	-
$n11$	9	-	-	-	-	-	-	-	-	-	▨	-	-	-	-	-	-	10
$R1$	-	-	-	6	-	-	-	-	-	-	-	▨	-	-	-	-	-	-
$R2$	-	-	-	-	-	5	-	-	-	-	-	-	▨	13	-	-	-	-
$R3$	-	-	-	-	-	-	6	-	-	-	-	-	-	▨	7	-	-	-
$R4$	-	-	-	7	-	-	-	6	-	-	-	-	-	-	▨	-	-	-
$M5$	-	7	-	-	-	-	-	-	-	-	6	-	-	-	-	▨	-	-
$R6$	12	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	▨	13
$R7$	-	6	-	-	-	-	-	-	-	-	7	-	-	-	-	-	-	▨

Table 4.1: Transportation time between different nodes of the FMS

Five basic components (plate, axis comp, I comp, r comp, L comp, and screw comp) are assembled with different ways to form four products in the shape of letters (B, E, L and T) (Figure 4.3). Each product formation requires several operations: loading plate on the shuttle, different assembly operations, an inspection of the product, and finally unloading product.

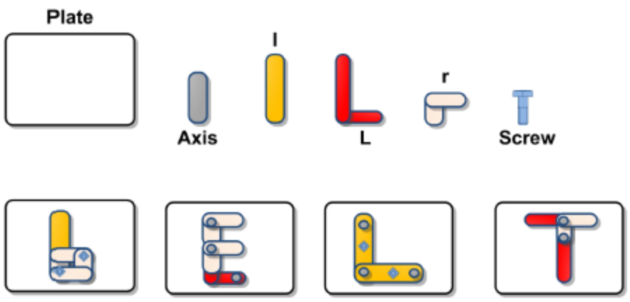


Figure 4.3: Basic components and products

Table 4.2 depicts different operations, their processing time, and possible machine in which an operation is performed.

Operation name	**Designation**	**Processing time (s)**	**Realized by**
Op_1	Plate loading	10	$R1$
Op_2	Axis mounting	20	$R2, R3$
Op_3	r_ comp mounting	20	$R2, R3$
Op_4	L_ comp mounting	20	$R2, R4$
Op_5	I_ comp mounting	20	$R4$
Op_6	Screw_ comp mounting	20	$R3, R4$
Op_7	Inspection	5	$M5$
Op_8	Plate unloading	10	$R1$

Table 4.2: Elementary operations (processing time, possible affectation)

In addition, the sequences of fabrication for each product are depicted in Table 4.3 Table 4.3 is read vertically for each product type column. For example, a product of type B is produced by executing the ordered sequence: once Op_1, three times Op_2, twice Op_3, once Op_4, once Op_6, once Op_7 and finally Op_8 once.

Operation name	**Product types and assembly sequence**			
	B	E	L	T
Op_1	1	1	1	1
Op_2	3	3	3	2
Op_3	2	2	-	1
Op_4	1	-	2	-
Op_5	-	1	-	1
Op_6	1	-	2	-
Op_7	1	1	1	1
Op_8	1	1	1	1

Table 4.3: The sequence of fabrication for each product

4.1.2 The proposed Multi-agent system (MAS)

The proposed MAS represents a set of soft agents, which describe the physical system entities (products, machines and routings). The different agents interact and cooperate to achieve the desired control functions. The description of different kind of agents is as follows:

(a) **Product agent (AgP)**:

- Each AgP represents an available product in the system.
- The agent has an overview of product status as processing time of each operations, operation carried, destination machine, etc.
- AgP is responsible to introduce the PDS paradigm by the product itself as a physical aspect and it can apply its intelligence as a decisional aspect.

(b) **Machine agent (AgM)**:

- Each AgM represents an available robot machine in the assembly
- The agent has an overview of machine status like machine queue status, products registered on the machine and the processing time of the operation.
- It has the responsibility to allow the product to pass to the corresponding machine or not.

(c) **Routing agent (AgR)**:

- It represents the turntable of the physical assembly as decision point
- It determines the product travel in the system.

4.1.3 Communication protocol of the proposed MAS

Two types of messages are possible between different agents in order to form the MAS communication protocol: registration messages and routing messages.

(a) **Registration message:**

- This happens at beginning of the next operation of the product will be performed in a machine.
- AgP sends information messages to different appropriate machine agents (AgMs) to inquire about machine status.
- AgMs send back corresponding machine status to AgP when it selects the appropriate machine according to its rule.
- AgP sends a registration message to the selected AgM.

(b) **Routing message**: The routing nodes are decisional points where the decision is made to define the product path through the system. According to the type of routing, there are two kinds of possible routing messages:

- *Product in front of machine rooting nodes*

  - AgP sends routing request message to AgM.
  - The travel of product on the machine is authorized or not authorized depending on machine rule and its queue status.
  - AgR receives a routing order message from AgM to realize the approved routine.
- *Product in front of ordinary rooting*
  - AgP sends a routing order message to AgR in order to adjust the appropriate routing according to product status.

Table 4.4 summarize all the messages used in this proposed for both ACL that ensures inter-agent communication [18], and the communication between agents and their corresponding physical entity with TCP/IP protocol.

Positioning product	**Message type**	**Transmitter→reciver**	**Designation**
Output of machine node (operation is finished)	TCP/IP	PVirtual[1]→AgP	Looking to accomplish next operations.
	ACL	AgP→AgMs	Machines status request (information).
		AgMs→AgP	Machines status send back (information)
		AgP→AgM	Registration on machine destination (order)
Machine routing node	TCP/IP	PVirtual→AgP	Looking for appropriate routing.
	ACL	AgP→AgM	Routing message (request)
	TCP/IP	AgM→M	Queue machines status request (information).
		M→AgM	Queue machines status send back (information).
	ACL	AgM→AgR	Routing message (order).
	TCP/IP	AgM→M	Diffusion of operation processing time.
		AgR→R	Diffusion of appropriate routing.
Ordinary routing node	TCP/IP	PVirtual→AgP	Looking for appropriate routing.
	ACL	AgP→AgR	Routing message (order).
	TCP/IP	AgR→R	Diffusion of appropriate routing.

Table 4.4: Exchanged messages within MAS into DES model

[1]PVirtual refers to the virtual product of simulation.

4.1.4 Product-driven approach

The employment of MAS offers to us the possibility to develop an intelligent product. The implementation of the agents can make a product the main actor in decision-making process throughout its life cycle in the shop floor.

AgP allows the product to apply intelligence, which is effective in the decision making of FMS by sharing this intelligence with the aid of infotronic technology (RFID, WIFI, etc.). [43]

Product is represented by unique identification (RFID tag), it is localized anywhere in the system (depending on RFID reader location) and it can store its data and communicate with its environment [44, 45]

4.1.5 FMS scheduling problem

Some operations are performed by different machines and there are different ways to tranfer jobs between machines; what introduce the flexibility of the system. Inorder to resolve our scheduling problem, Priority dispatching rules (PDRs) based Product-driven system are proposed to control the robotised assembly cell.

PDRs are mainly used to face a dynamic unpredictable environment. They are able to determine the processing priority of a job among several waiting jobs, on a maching during the manufacuring process.

The proposed approach is integrated into a Multi-agent Control Framework.The scheduling function is decomposed and distributed on different AgPs and AgMs. Accordingly, each product available in the system can select a machine according to five proposed product PDR's assigned to product agent (Table 4.6). In addition, every machine performs the passage order of registered product according to four proposed machine PDRs assigned to each machine agent (Table 4.5).

PDRs machine name	**Description**
SPT	A product which has the shortest processing time .
LPT	A product which has the longest processing time.
FIFO	First input first output.
LIFO	Last input first output.

Table 4.5: Dispatching rules assigned to a product

PDRs product name	**Description**
M_ SPT	The machine with where the smallest processing time of the operation is realized.
M_ LPT	The machine with where the largest processing time of the operation is realized.
RP1	The least loaded machine and the shortest transportation time between a product location and product destination.

Table 4.6: Dispatching rules assigned to a machine

4.2 Experimental set-up

After discussing in details the different behaviours of the studied FMS of AIP-PRIMECA and proposing a fitting PDS structure within a MAS framework; now, it is time to bring all those predefined paradigms and tools into practice, that is implementing the cyber-physical AIP-PRIMECA FMS.

4.2.1 DES model development

The DES model of AIP-PRIMECA FMS is developed in Flexsim 3D which provides a realistic appearance of objects (Figure 4.4).

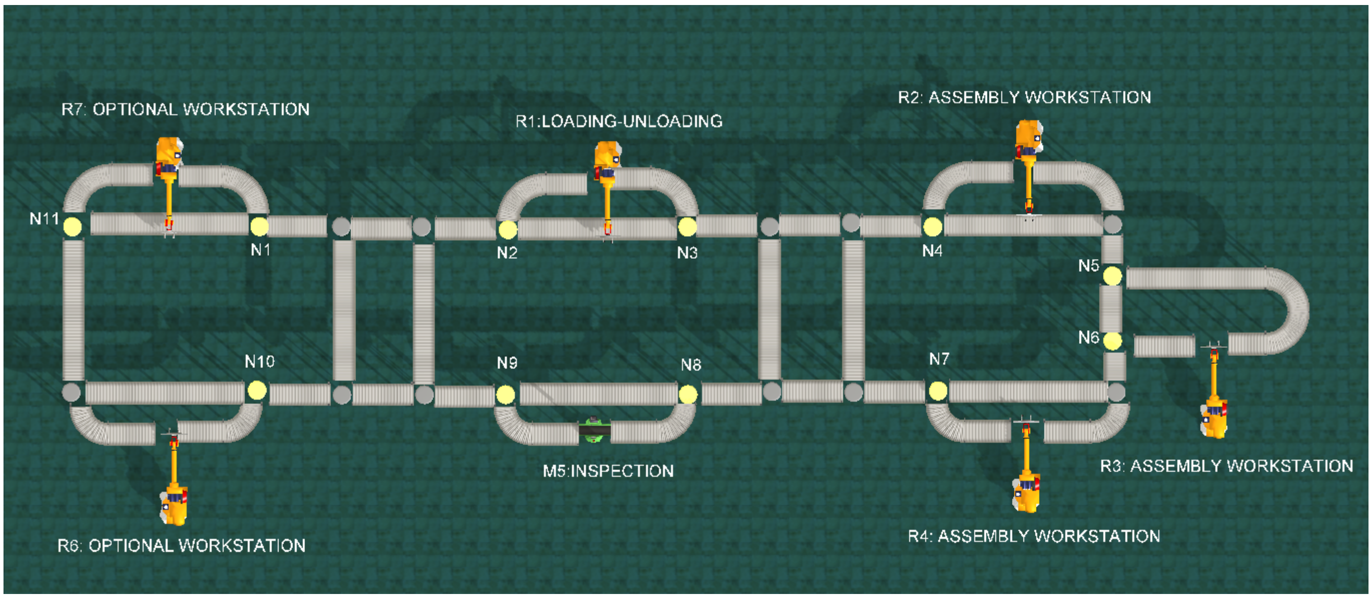


Figure 4.4: FlexSim DES model of AIP PRIMECA FMS

Before proceeding to different tests, it is necessary to have a valid and accurate model in the HILS to maintain the reliability and stability of the proposed DCS.
The DES model is verified and validated according to transportation time between different nodes of system (Table 4.1).

4.2.2 MAS development

The proposed MAS is developed on Netbeans IDE (see appendix A). It consists of distinct agents are connected with their corresponding virtual object through a sockets TCP/IP network.

- Five AgMs are created to represent the different robots in the system (simulation), ($R6$ and $R7$ are not included).
- Eight AgQs represent all the possible decision points (routing) in the system ($n1$,$n10$ and $n11$ not included).
- The number of AgPs depends automatically on the number of available product in the system.

The different exchanged messages are displayed in string format and they convey distinct information according to the message type. All this information is first coded by the sender agent and sent to the receiver agent. The receiver agent collects the sent information before their treatment.

Examples on sending messages between agents :

- **1z1z1z1z0z0z0z**:
  This message indicates that the available products in the system is 4 which are “B”, “E”,”L”, “T” and the rest are not available (z is separator).

-
  ```
  String m="load"+"z"+10+"z"+t[2]+"z";
  Send(m, "AgM1");
  ```
  This message indicates the agent t[2] (AgProdB) sends message to receiver agent agM1 to load the product B through it with processing time equal to 10 (z here is separator).

-
  ```
  String m = "routingQ1"+"z"+"1"+"z"+t[2]+"z";
  Send(m, "AgQ1");
  ```
  It is **a product in front of ordinary routing message** sent by product agent to decision point *Q1* to rotate (open), 1 indicates the inspection=0 and affectation=1.

- The registration message sent by an AgP to the selected AgM. The character “z” is designed as a separator between diferent conveyed information and the sign “w” is designed to specify the message ends (**Registration z processing time z AgP w**). For instance:

  ```
  String m = "timing_M2"+"z"+"Prod "+t[1]+"z"+s11+ "w";
  SendSocket(m);
  ```

  This message holds the processing time duration of product **Prod t[1]** which is product **B** in **the Robot M2** and sent to flexsim simulation to recover **the duration**, at same time save this message on history table as: **timing_M2zProd B1z100**, where processing time is 100.

4.2.3 HILS validation strategy

A virtual product keeps circulating in the virtual system (DES model) until it reaches a decisional point where it solicited HILS mechanism. Then, HILS mechanism involves the physical product in decision making. After that, the corresponding physical product allows the virtual product to trigger its correspondig AgP within MAS. Finally the decision taken is applied in the DES model.

There are several decisional point where an RFID reader cn be positioned. The choice of a the decisional point is crucial to avoid redundancy of the test and hence improve its cost-effectiveness. To achieve the desired test, a strategy is developed. It relies on the following:

- Several decisional points are identified in the system and can contribute in validating the PDS.
- The identified decisional points are distinguished into three groups:
  - Product in output of machine nodes ($R1$, $R2$, $R3$, or $R4$).
  - Product in front of machine routing nodes ($n2$, $n4$, $n5$, $n6$ or $n8$).
  - Product in front of ordinary routing nodes ($n3$, $n7$ or $n9$).
- Decision making are the same in decision points belonging to the same group (the same agent interaction mechanism for each group). Consequently, a single decisional point represents fairly the group of decisional points to which it belongs. We select for instance:
  - The decisional point in the output of the machine $R2$.

– The decisional point in front of the ordinary routing node *n*3. The decision point in front of the machine routing node *n*4.

According to Table 4.4, the decisional mechanisms employed in decisional point *n*4 are the most complex among the three chosen decisional points. The three types of agents (AgP, AgM and AgR) are solicited in decision making when a product is in front of *n*4 (Figure 4.5). Furthermore, the choice of the decisional point *n*4 is a good choice to highlight the distributed control of dynamic routing processes for intelligent products.

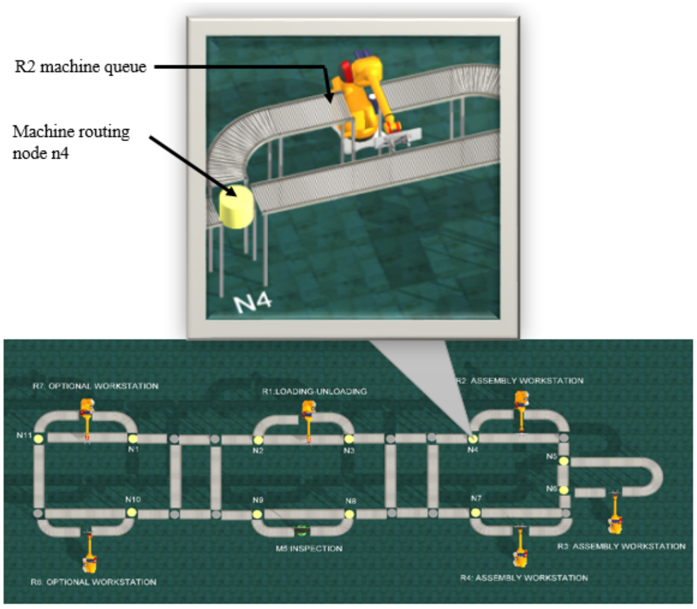


Figure 4.5: Zoom-in the selected decisional point *n*4

4.2.4 Test procedure

The aim of the proposed HILS validation test, the synchronization between the virtual process model and the physical process. For this, the turntable is used as a physical automated mean of synchronization between virtual flow and physical flow. The following strategy is adopted for the test:

1. Concerning the simulation loop:
    (a) The launch of a production line of four (4) virtual products in the DES model (Types B, E, L and T respectively).
    (b) The queue capacity of each machine is fixed to one single product (no more than one pending product in a machine queue).
    (c) FIFO PDR is the selected for all machines.
    (d) M_SPT PDR is selected for the four products.

(e) Successive assembly operations are carried out in the same machine on the present product.

2. Concering the hardware elements

    (a) Real products (four colored Legos) associated with each virtual product.

    (b) Each real product is tagged by RFID with a UID (Unique ID).

    (c) Arduino Uno μC board is used to command the RFID tag reading through RC522 RFID module. The μC board commands also the turntable rotation.

    (d) Siemens ET 200SP PLC[1] is used to command the virtual flow depending on the RFID reading.

Four AgPs are created and associated with the four virtual products we have launched (types: B, E, L and T). Once each product loading operation is completed (at the output of R1), the virtual product is looking for the destination machine where its next operations will be realized. When the virtual product arrives in front of the routing node $n4$, it activate a boolean PLC tag via OPC UA, and then, HILS procedure is initiated. The procedure progresses as follows: (see Figure 4.6)

Step 1: Via OPC UA, the DES system writes the present virtual product type (1, 2 ,3 or 4 corresponding to B, E, L,or T respectively) in a PLC Tag.

Step 2: Via Modbus TCP, the PLC tag passes the type to Arduino μC board.

Step 3: According to that received type and the physical tag UID reading by RC522 RFID module, Arduino rotates the turntable until bringing the corresponding physical product type at the RC522 position.

Step 4: Via Modbus TCP, Arduino sends the physical product type to the ET 200SP PLC.

Step 5: **If** the corresponding physical product type received, **then** a boolean PLC tag is activated and the DES system read it via OPC UA. **Else**, **go to** step 3.

Step 6: Via TCP/IP, the virtual product contacts its corresponding agent (AgP) and resume its routing in the DES system.

[1]A simulated ET 200SP is used instead of a real physical one which was unavailable as a result of training cancellation. PLCSIM Advanced software (see appendix C) simulates almost all functionalities of a real ET 200SP PLC

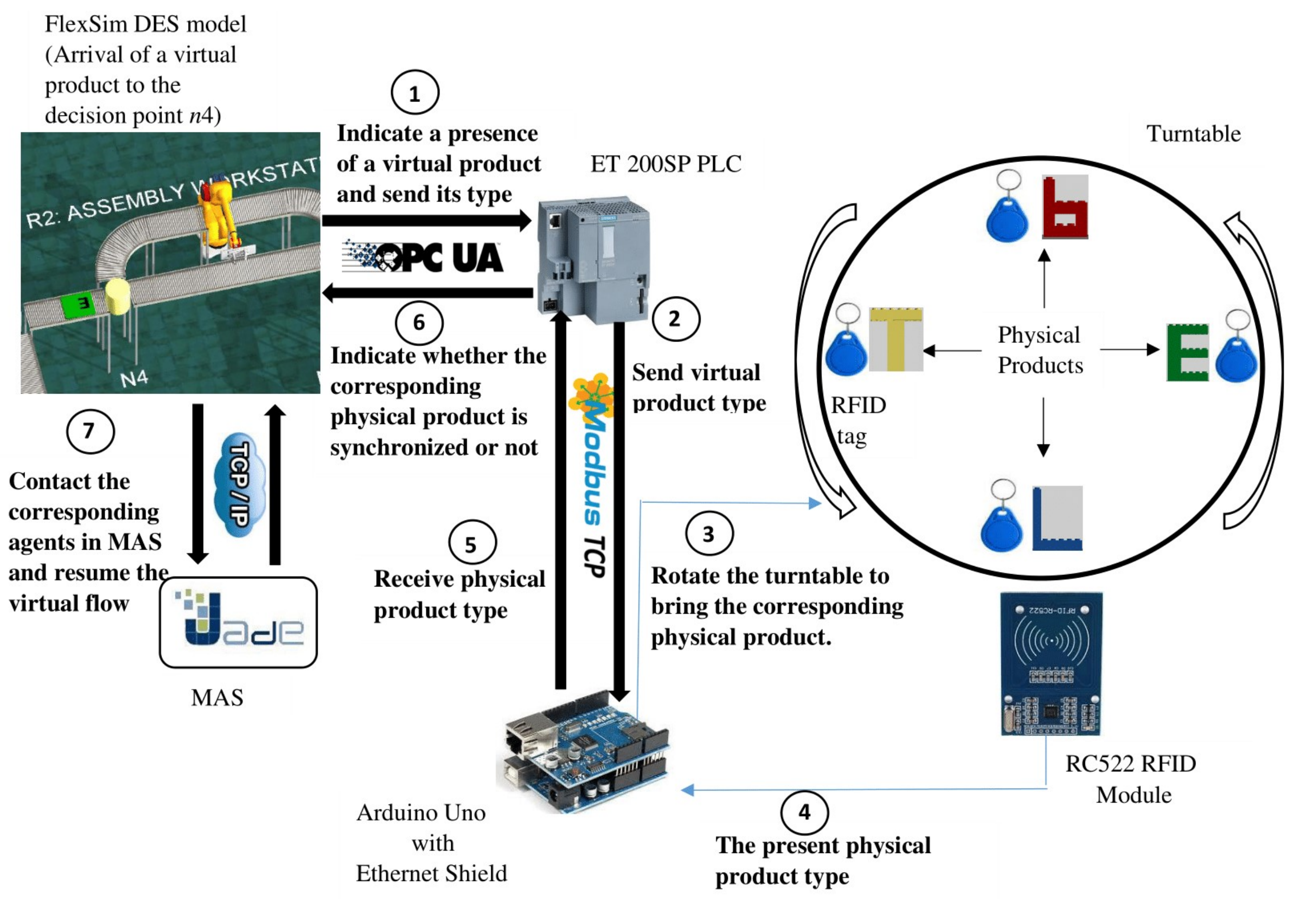

FlexSim DES model (Arrival of a virtual product to the decision point n4)
R2: ASSEMBLY WORKSTAT
N4
1
Indicate a presence of a virtual product and send its type
OPC UA
ET 200SP PLC
6
Indicate whether the corresponding physical product is synchronized or not
2
Send virtual product type
Modbus TCP
Turntable
Physical Products
RFID tag
7
Contact the corresponding agents in MAS and resume the virtual flow
TCP/IP
Jade
MAS
5
Receive physical product type
3
Rotate the turntable to bring the corresponding physical product.
RC522 RFID Module
4
The present physical product type
Arduino Uno with Ethernet Shield

Figure 4.6: HILS validation scenario

4.2.5 Results and discussion

According to M_SPT rule (Table 4.6), AgPs associated to products B and L select machine *R*2 to perform the next sequence of assembly operations($3Op_22Op_3$ and $3Op_2$ respectively), whereas, AgPs associated with products E and T select machine *R*3 the corresponding next sequence of assembly operations ($3Op_22Op_3$ and $2Op_2Op_3$ respectively).

Since FIFO is selected for both machines, two situations occurs:

- Product B access firstly to machine *R*2 and product L access to machine R2 queue pending for completion of product B operations.
- Product E access to machine R3 and product T access to machine R3 queue pending for completion of product E operations.

Figure 4.7 is a Gantt chart showing the time schedule of assembly sequence in machines *R*2 and *R*3.

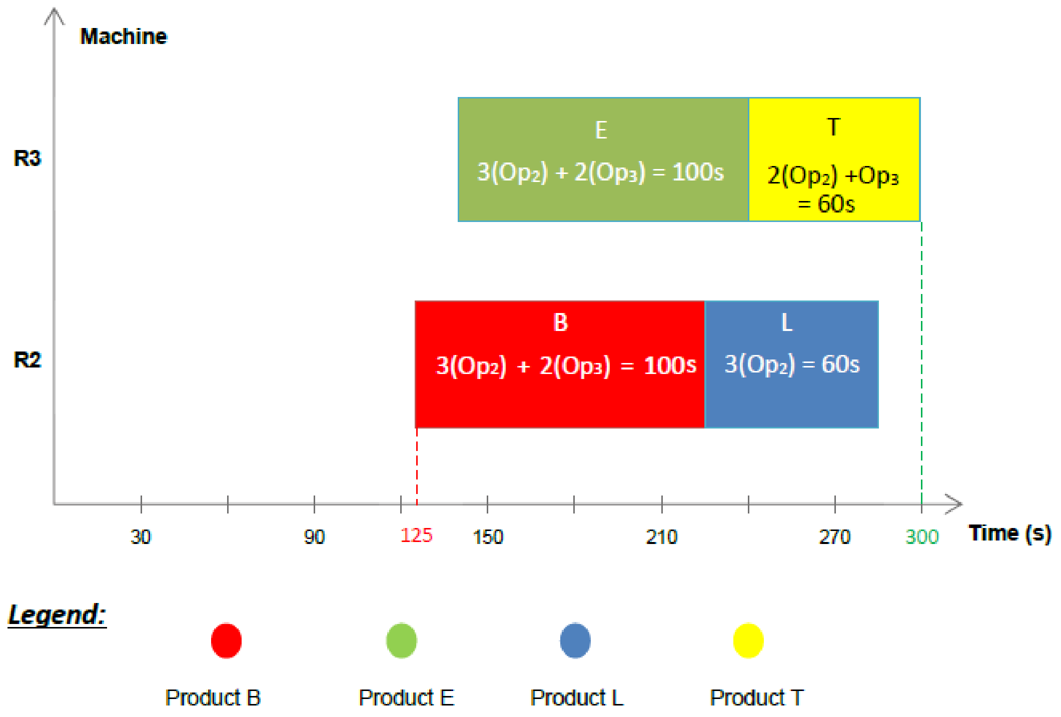
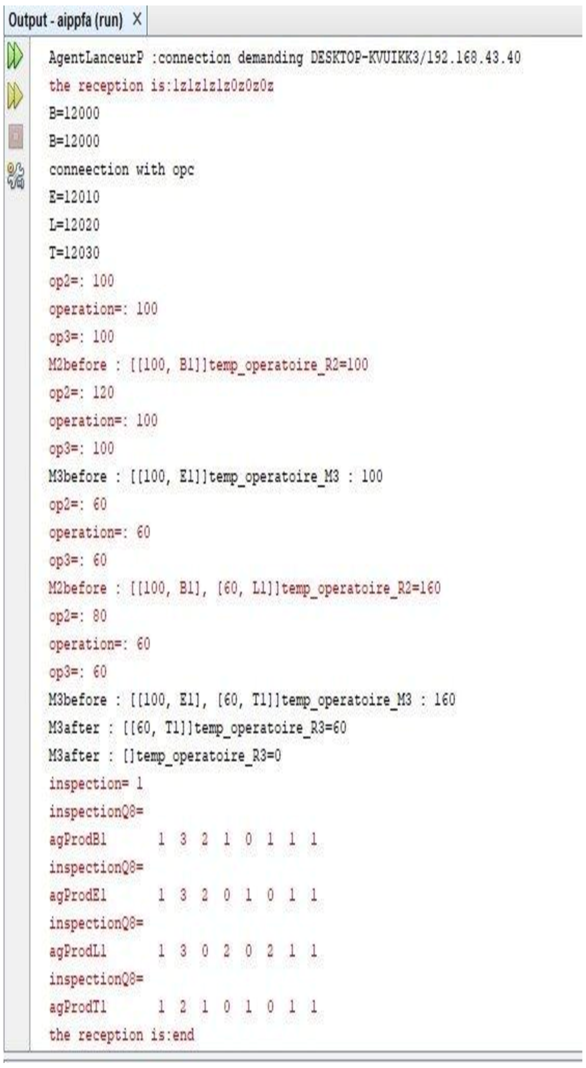


Figure 4.7: Gantt chart of *R*2 and *R*3 machines opeations

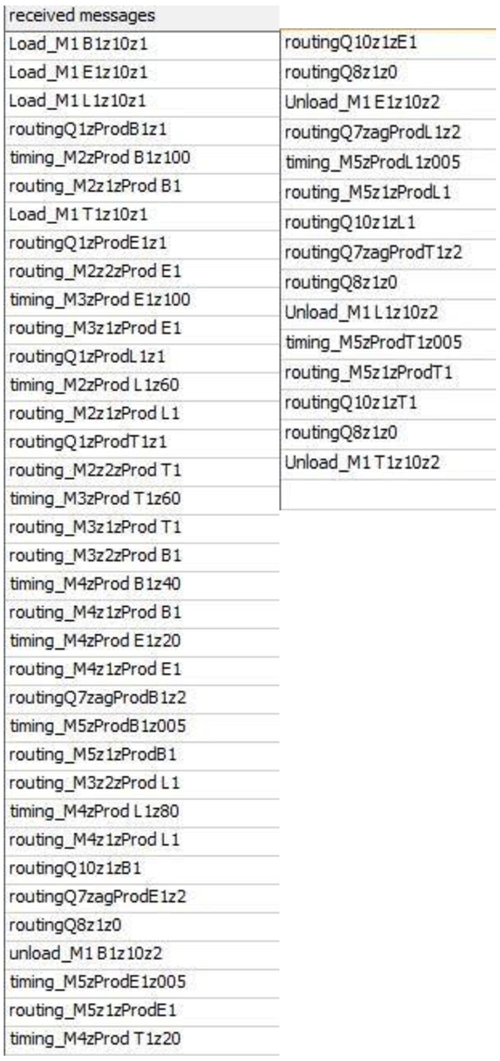
Synchronization time induced by HILS mechanism (time taken by the turntable to bring the corresponding physical product) does not affect the objectives of this validation technique in this study. Accordingly, it is not mentioned in the Gantt chart of Figure 4.7.

Control messages sent by the MAS to the DES model are stored in a history table in FlexSim. Figure 4.8b shows that messages are properly during all products cycles. The output of MAS control application (Figure 4.8a) shows the assembly sequence of each of the four products, it matches exactly the sequences defined previously in Table 4.3.



```
Output - aippfa (run)
AgentLanceurP :connection demanding DESKTOP-KVUIKK3/192.168.43.40
the reception is:1z1z1z1z0z0z0z
B=12000
B=12000
conneection with opc
E=12010
L=12020
T=12030
op2=: 100
operation=: 100
op3=: 100
M2before : [[100, B1]]temp_operatoire_R2=100
op2=: 120
operation=: 100
op3=: 100
M3before : [[100, E1]]temp_operatoire_M3 : 100
op2=: 60
operation=: 60
op3=: 60
M2before : [[100, B1], [60, L1]]temp_operatoire_R2=160
op2=: 80
operation=: 60
op3=: 60
M3before : [[100, E1], [60, T1]]temp_operatoire_M3 : 160
M3after : [[60, T1]]temp_operatoire_R3=60
M3after : []temp_operatoire_R3=0
inspection= 1
inspectionQ8=
agProdB1      1  3  2  1  0  1  1  1
inspectionQ8=
agProdE1      1  3  2  0  1  0  1  1
inspectionQ8=
agProdL1      1  3  0  2  0  2  1  1
inspectionQ8=
agProdT1      1  2  1  0  1  0  1  1
the reception is:end
```

(a) Java MAS application output



```
received messages
Load_M1 B1z10z1
Load_M1 E1z10z1
Load_M1 L1z10z1
routingQ1zProdB1z1
timing_M2zProd B1z100
routing_M2z1zProd B1
Load_M1 T1z10z1
routingQ1zProdE1z1
routing_M2z2zProd E1
timing_M3zProd E1z100
routing_M3z1zProd E1
routingQ1zProdL1z1
timing_M2zProd L1z60
routing_M2z1zProd L1
routingQ1zProdT1z1
routing_M2z2zProd T1
timing_M3zProd T1z60
routing_M3z1zProd T1
routing_M3z2zProd B1
timing_M4zProd B1z40
routing_M4z1zProd B1
timing_M4zProd E1z20
routing_M4z1zProd E1
routingQ7zagProdB1z2
timing_M5zProdB1z005
routing_M5z1zProdB1
routing_M3z2zProd L1
timing_M4zProd L1z80
routing_M4z1zProd L1
routingQ10z1zB1
routingQ7zagProdE1z2
routingQ8z1z0
unload_M1 B1z10z2
timing_M5zProdE1z005
routing_M5z1zProdE1
timing_M4zProd T1z20
routingQ10z1zE1
routingQ8z1z0
Unload_M1 E1z10z2
routingQ7zagProdL1z2
timing_M5zProdL1z005
routing_M5z1zProdL1
routingQ10z1zL1
routingQ7zagProdT1z2
routingQ8z1z0
Unload_M1 L1z10z2
timing_M5zProdT1z005
routing_M5z1zProdT1
routingQ10z1zT1
routingQ8z1z0
Unload_M1 T1z10z2
```

(b) History of received messages in FlexSim

Figure 4.8: Exchanged messages between MAS and DES

Conclusion

This chapter presented the implementation of a CPS emulating AIP-PRIMECA FMS relying on theoretical knowledge acquired from previous chapters. A MAS Java application is designed to control the DES model of the FMS based on product-driven approach. HILS validation technique (including RFID technology, PLC and Arduino μC board) is applied to a selected decisional point. The approach has shown promising results.

General Conclusion

To conclude our work, we can say that it was an interesting challenge to implement a cyber-physical manufacturing system based on PDS paradigm, controlled with a MAS Java application and validated with HILS technique.

In the theoretical part, we focused on the concepts, theories of all the approaches, techniques, protocols, software and hardware used in this study.

The generalities on simulation are firstly presented as well as the choice of FlexSim software for simulating industrial systems whether for educational purposes or for an industry projects.

Next, we represented a theories of a different techniques used to link the cyberspace (DES model and JADE MAS) with the physical world (Arduino, RFID, routing or turntable) using Sockets, OPC UA and Modbus protocol, also we concentrated about more detailed information on the hardware used (PLC, Arduino, RFID) to close the loop of HILS approach.

In the practical part, the proposed MAS is implemented using Java program, distinct smart agents are also created representing different active parts of the system (products, robots, routings). They are provided by priority dispatching rules as intelligent decision contribution, sharing information between them, communicating with emulator via socket TCP/IP as well as commanding their different virtual parts.

The emulator is designed by Flexsim software that contains the 3D simulation of AIP-PRIMECA, many program functions for communicating with other external softwares as JADE via Sockets and PLCSIM via OPC UA and different codes for managing and ordering the sent messages from the MAS.

RFID technology is used to ensure real time communication between the hardware and software. The employment of this technology helped the HILS approach to be easily used for the V&V of the proposed DCS.

Although all the problem that have faced, especially some communication and hardware malfunctions, the main objective was successfully achieved by combining all those approaches together to reach so called hardware in the loop simulation for product driven control of cyber-physical manufacturing system.

As further work, we propose:

- Expand our Hardware loop simulation by inserting more hardware as conveyor, robot which means more additional protocols, algorithms and costs.
- Merge the dispatching rules method with optimization algorithms as neural network, search harmony methods to minimize makespan or energy.
- Back up our communication between the duo spaces by IOT by inserting UDP protocol, WIFI, web platform development.

Appendix A

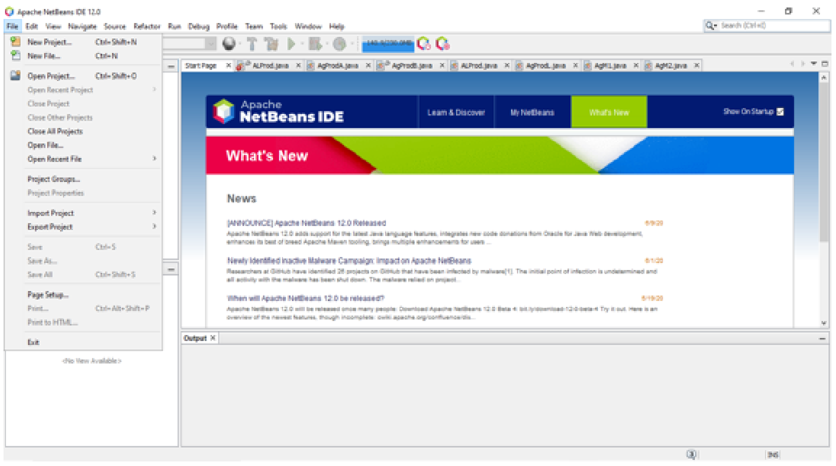
NetBeans IDE

NetBeans is an Integrated Development Environment (IDE) for Java. NetBeans allows applications to be developed from a set of modular software components called modules. NetBeans runs on Windows, macOS, Linux and Solaris. In additional to Java development, it has extensions for other languages like PHP, C, C++, HTML5 and JavaScript. NetBeans IDE supports development of all Java application types (Java SE, Java ME, web, EJB and mobile applications).[46]

A.1 Project creation

When you create an IDE project, you create an environment in which to build and run your applications. Using IDE projects eliminates configuration issues normally associated with developing on the command line. You can build or run your application by choosing a single menu item within the IDE in the following manner :

1. Start NetBeans IDE.
2. In the IDE, choose File, New project as shown in Figure A.1 below.



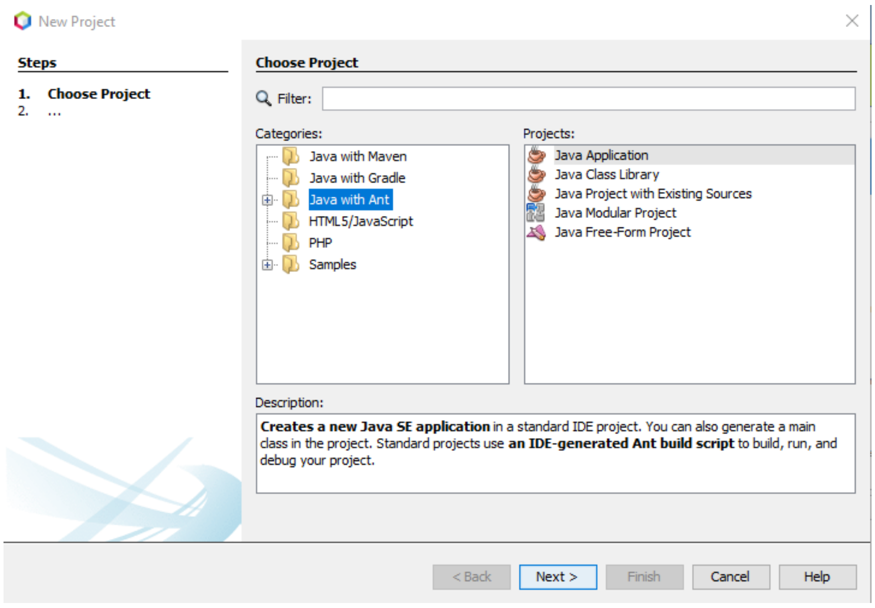
[ANNOUNCE] Apache NetBeans 12.0 Released 6/9/20
Apache NetBeans 12.0 adds support for the latest Java language features, integrates new code donations from Oracle for Java Web development, enhances its best of breed Apache Maven tooling, brings multiple enhancements for users ...

Newly Identified Inactive Malware Campaign: Impact on Apache NetBeans 6/1/20
Researchers at GitHub have identified 26 projects on GitHub that have been infected by malware[1]. The initial point of infection is undetermined and all activity with the malware has been shut down. The malware relied on project...

When will Apache NetBeans 12.0 be released? 5/19/20
Apache NetBeans 12.0 will be released once many people: Download Apache NetBeans 12.0 Beta 4: bit.ly/download-12-0-beta4 Try it out. Here is an overview of the newest features, though incomplete: cwiki.apache.org/confluence/dis...

Figure A.1: Windows platform of NetBeans software

3. In the new project wizard expand the java category and select Java application as shown in FigureA.2 below, then click Next.

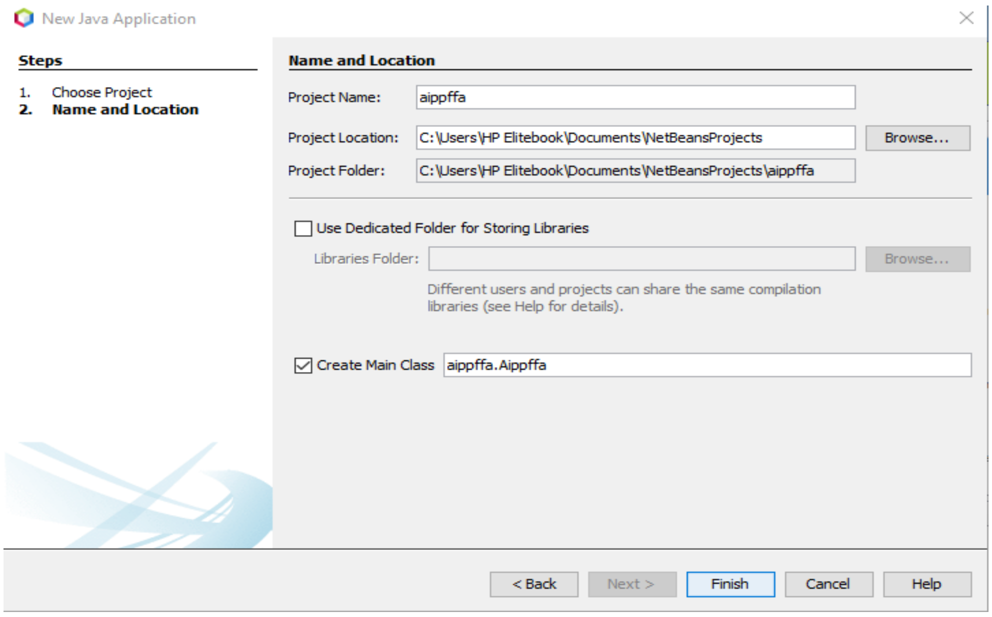


Creates a new Java SE application in a standard IDE project. You can also generate a main class in the project. Standard projects use **an IDE-generated Ant build script** to build, run, and debug your project.

Figure A.2: New Project wizard, choose Project page

4. In the Name and Location page of the wizard (Figure A.3), do the following:

   - In the project Name field, type your project's name (”aippffa” for our project).
   - In the Create Main Class field, type your main class(aippffa.aippffa or like our main class which is:jade.Boot).
   - Finally, click finish.



Project Name: aippffa

Project Location: C:\Users\HP Elitebook\Documents\NetBeansProjects

Project Folder: C:\Users\HP Elitebook\Documents\NetBeansProjects\aippffa

☐ Use Dedicated Folder for Storing Libraries

Libraries Folder:

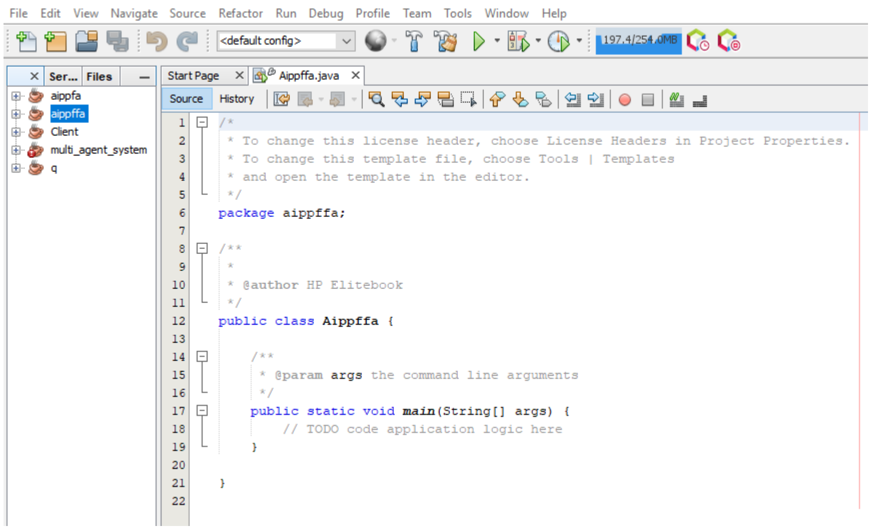
Different users and projects can share the same compilation libraries (see Help for details).

☑ Create Main Class aippffa.Aippffa

Figure A.3: New Project wizard, name and location page

5. The project is created and opened in the IDE. The essential components should be seen as follow:

   - The Project's window, which contains a tree view of the components of the project, including source files, libraries that your code depends on, and so on.
   - The Source Editor window with a file called "aippffa.java" open.
   - The Navigator window, which you can use to quickly navigate between elements within the selected class. (see Figure A.4)



```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template editor.
 */
package aippffa;

/**
 *
 * @author HP Elitebook
 */
public class Aippffa {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
    }

}
```

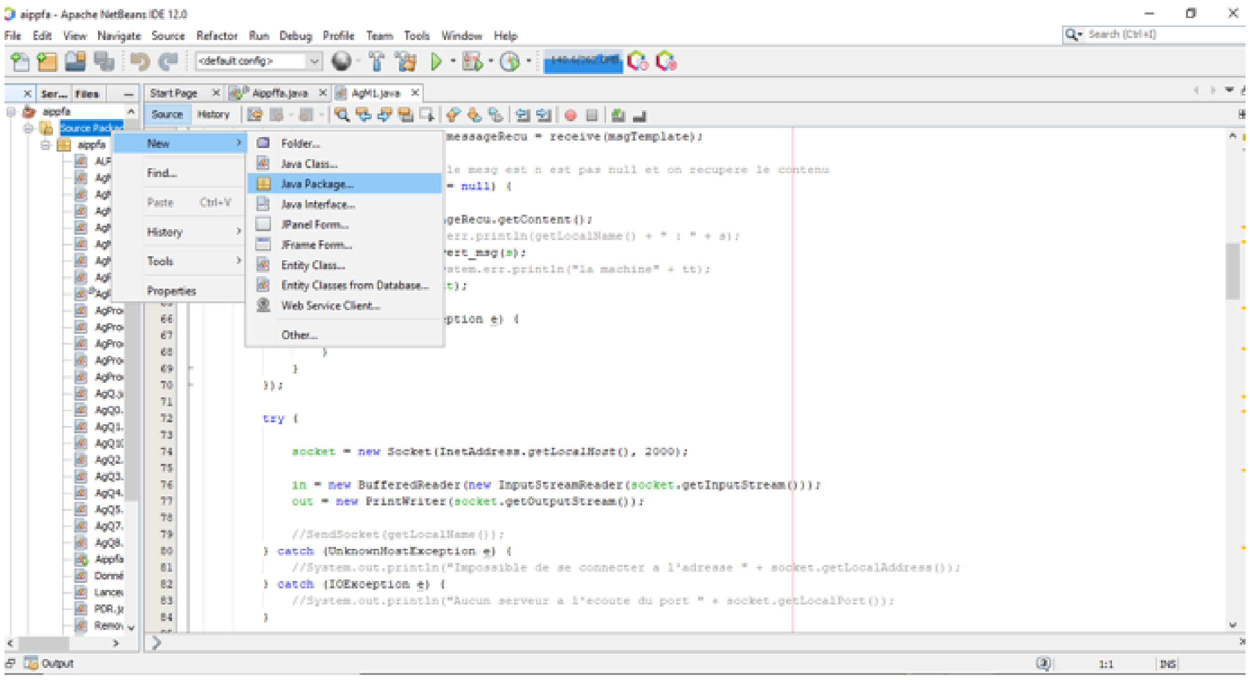
Figure A.4: NetBeans IDE with the Aippffa project open

A.2 Class creation

To create Java class in NetBeans, you should follow all the steps:[46]

1. Create a new Java application. Be sure to uncheck Create main class.
2. Right-click on the Source Packages folder and select New, Java Package. Enter your webmail user name for the name of the project.
3. Right-click the user name package and select New, Java Main Class.
4. Name your class menu.
5. Run the project. You will be prompted to select the main class.

Figure A.5 summarize those steps:



```
messageRecu = receive(msgTemplate);

le mesg est n est pas null et on recupere le contenu
= null) {

geRecu.getContent();
err.println(getLocalName() + " : " + s);
ert_msg(s);
stem.err.println("la machine" + tt);
t);

ption e) {

            }
        }
    });

    try {

        socket = new Socket(InetAddress.getLocalHost(), 2000);

        in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
        out = new PrintWriter(socket.getOutputStream());

        //SendSocket(getLocalName());
    } catch (UnknownHostException e) {
        //System.out.println("Impossible de se connecter a l'adresse " + socket.getLocalAddress());
    } catch (IOException e) {
        //System.out.println("Aucun serveur a l'ecoute du port " + socket.getLocalPort());
    }
```

Figure A.5: Java class creation in NetBeans IDE

Appendix B

TIA Portal

The Totally Integrated Automation Portal, referred to as TIA Portal in the following, offers all the functions you need for implementing your automation task assembled in a single, cross software platform. It is the first shared working environment for integrated engineering with the various SIMATIC systems made available within a single framework. The TIA Portal therefore also enables reliable, convenient cross-system collaboration for the first time. All required software packages, from hardware configuration and programming to visualization of the process are integrated in a comprehensive engineering framework.[47]

B.1 Advantages of working with TIA Portal

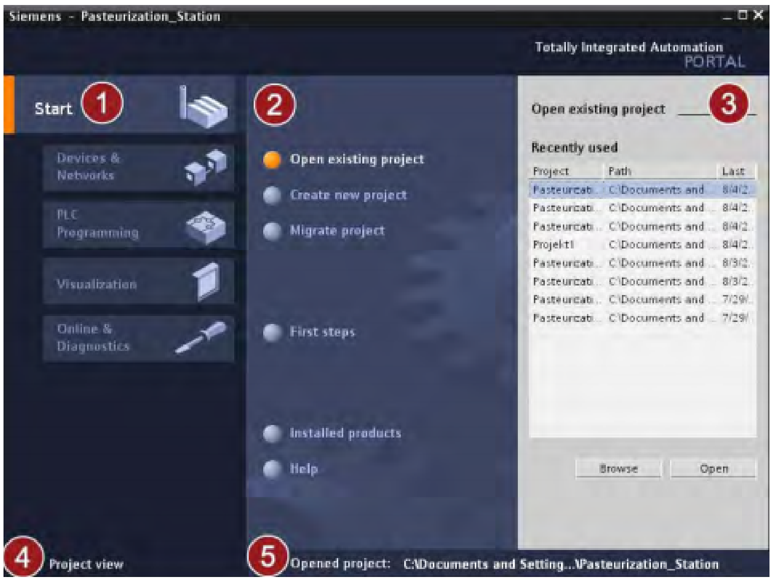
The following features provide efficient support during the realization of your automation solution when working with TIA Portal:

- Integrated engineering with a uniform operating concept.
- Consistent, centralized data management with powerful editors and universal symbols.
- Comprehensive library concept.
- Multiple programming languages.

B.2 User Interface

B.2.1 Portal view

The portal view provides a task-oriented view of the toolbox. The goal of the portal view is to provide a simple navigation in the tasks and data of the project. This means the functions of the application can be reached via individual portals for the most important tasks. Figure B.1 shows the structure of the portal view:



Project	Path	Last
Pasteurizati...	C:\Documents and ...	8/4/2...
Pasteurizati...	C:\Documents and ...	8/4/2...
Pasteurizati...	C:\Documents and ...	8/4/2...
Projekt1	C:\Documents and ...	8/4/2...
Pasteurizati...	C:\Documents and ...	8/3/2...
Pasteurizati...	C:\Documents and ...	8/3/2...
Pasteurizati...	C:\Documents and ...	7/29/...
Pasteurizati...	C:\Documents and ...	7/29/...

Figure B.1: The structure of TIA Portal platform [47]

① *Portals for the different tasks:* The portals provide the basic functions for the individual task areas. The portals that are provided in the portal view depends on the products that have been installed.

② *Actions for the selected portal:* Here, you will find the actions available to you in the portal you have selected. You can call up the help function in every portal on a context-sensitive basis.

③ *Selection panel for the selected action:* The selection panel is available in all portals. The content of the panel adapts to your current selection.

④ *Switch to project view:* You can use the ”Project view” link to switch to the project view.

⑤ *Display of the project that is currently open:* Here, you can obtain information about which project is currently open.

B.2.2 Project view

The project view is a structured view of all components of a project. In the project view the various editors are available that you can use to create and edit the corresponding project components.Figure B.2 shows the structure of the project view:

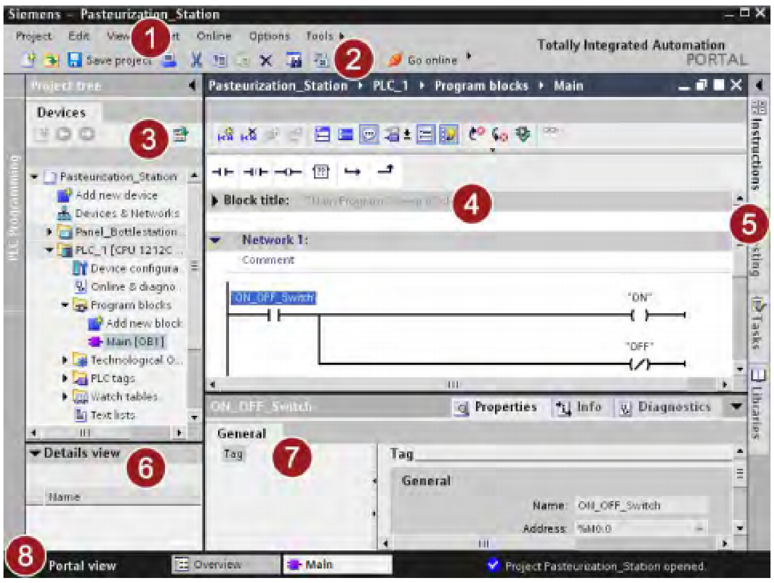


Figure B.2: TIA Portal project view [47]

① *Menu bar:* The menu bar contains all the commands that you require for your work.

② *Toolbar:* The toolbar provides you with buttons for commands you will use frequently. This gives you faster access to these commands than via the menus.

③ *Project tree:* The project tree gives you access to all components and project data.

④ *Work area:* The objects that you can open for editing purposes are displayed in the work area.

⑤ *Task cards:* Task cards are available depending on the edited or selected object. The task cards available can be found in a bar on the right-hand side of the screen. You can collapse and reopen them at any time.

⑥ *Details view:* Certain contents of a selected object are shown in the details view. This might include text lists or tags.

⑦ *Inspector window:* Additional information on an object selected or on actions executed are displayed in the Inspector window.

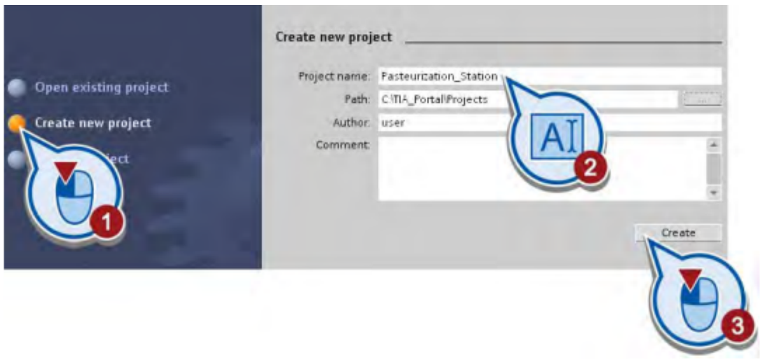
⑧ *Switching to portal view:* You can use the ”Portal view” link to switch to the portal view.

B.3 Project creation

The following steps show how to create a new project:[47]

1. Start Totally Integrated Automation Portal (TIA Portal).

2. Create the project under any path as shown in Figure **??**



Create new project

Project name: Pasteurization_Station
Path: C:\TIA_Portal\Projects
Author: user
Comment:

Create

Figure B.3: Tia Portal project creation: specifying name and path [47]

Next, you should add a new PLC to the project and configure its properties. To add a new device to the project, follow these steps:

1. Use the portal to add a new device as shown in Figure B.4:

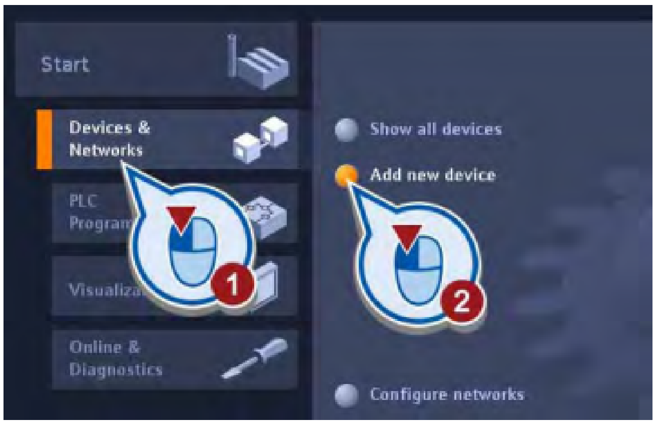
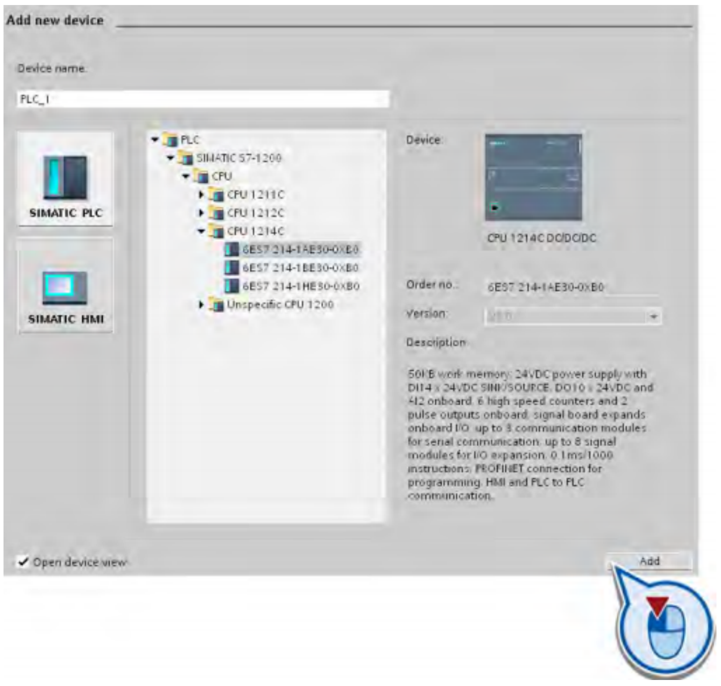


Figure B.4: Device & networks: add new device [47]

2. Select the desired PLC.

3. Make sure that the "open device view" option is enabled, if the option is not enabled, left click on the option to enable it.

4. Click "add" as shown in Figure B.5:



Device name: PLC_1

Device: CPU 1214C DC/DC/DC

Order no.: 6ES7 214-1AE30-0XB0

Version:

Description:

50KB work memory; 24VDC power supply with DI14 x 24VDC SINK/SOURCE, DO10 x 24VDC and AI2 onboard; 6 high speed counters and 2 pulse outputs onboard; signal board expands onboard I/O; up to 3 communication modules for serial communication; up to 8 signal modules for I/O expansion; 0.1ms/1000 instructions; PROFINET connection for programming, HMI and PLC to PLC communication.

Figure B.5: PLC device chosen [47]

With the PLC, the organization block “Main [OB1]” is automatically created in the project. The following steps show how to open the organization block in program editor:

1. Open “the program blocks” folder in the project tree.

2. Open the organization block “Main [OB1]” as shown in Figure B.6.

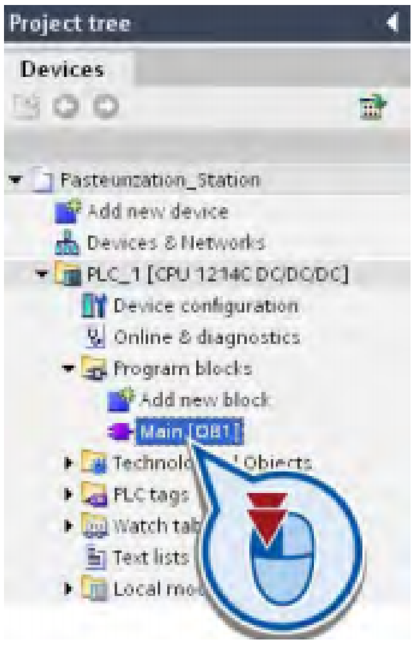


Figure B.6: Project tree, program blocks, Main [OB1] [47]

B.4 Tags

A tag is a variable used in the program that can take on different values. Depending on the range of application. The tags are divided into the following categories:

- *Local tags:* they apply only in the block in which they are defined.
- *PLC tags:* they apply throughout the entire PLC.

B.4.1 PLC tags

A PLC tags is made up of the following components:

- *Name:* The name of a tag is valid for a PLC and may only occur once within the entire program
- *Data type:* The data type defines the value representation and the permitted value range.
- *Address:* The address of a tag is absolute and defines the memory area from which the tag reads or writes a value.

Appendix C

S7-PLCSIM Advanced

Simulation systems support the development of programs and the deployment in production that follows. In the automation world, a simulated test environment shortens commissioning times. It is possible to test the program after program changes in the virtual controller before it is loaded into the corresponding real controller and the plant is put into operation.

Using S7-PLCSIM Advanced, we can simulate our CPU programs on a virtual controller without a need for a real controller for this. We can configure your CPU in TIA Portal, program our application logic and then load the hardware configuration and the program into the virtual controller. From there we can run our program logic, observe the effects of simulated inputs and outputs and adapt our programs. In addition to communicating via Softbus, S7-PLCSIM Advanced provides a full Ethernet connection and can thus also communicate distributed.[48]

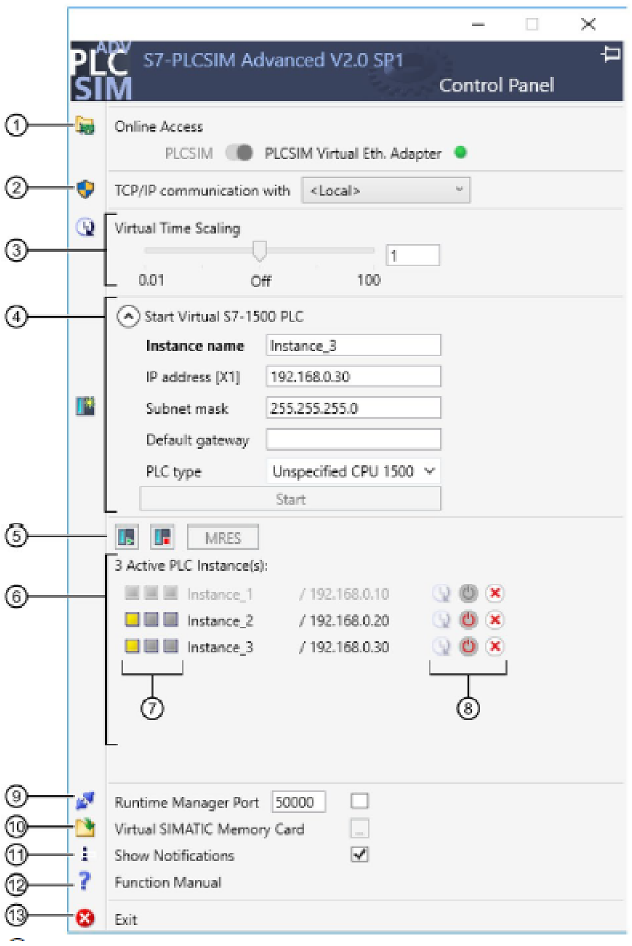
C.1 Advantages of the software

The use of S7 PLCSIM Advanced offers numerous advantages:

- Improve quality of automation projects.
- Accelerate time to market.
- Reduce production times.
- Reduce risk for commissioning.
- Avoid costs for hardware in simulation environments.
- Increase efficiency in maintenance.

C.2 User Interface

S7-PLCSIM Advanced provides a control panel (Figure C.1) for creating and operating instances of a virtual controller.[48]



Online Access
PLCSIM PLCSIM Virtual Eth. Adapter
TCP/IP communication with <Local>
Virtual Time Scaling 1
0.01 Off 100
Start Virtual S7-1500 PLC
Instance name Instance_3
IP address [X1] 192.168.0.30
Subnet mask 255.255.255.0
Default gateway
PLC type Unspecified CPU 1500
Start
MRES
3 Active PLC Instance(s):
Instance_1 / 192.168.0.10
Instance_2 / 192.168.0.20
Instance_3 / 192.168.0.30
Runtime Manager Port 50000
Virtual SIMATIC Memory Card
Show Notifications ✓
Function Manual
Exit

Figure C.1: PLCSIM Advanced control panel [48]

1. *Online Access*: Switch to select the communication interface (either local via softbus or via TCP/IP).

2. *TCP/IP communication*: Selection of network adapter for distributed communication.

3. *Virtual Time Scaling*: Slider to adjust the scaling factor.

4. *Start Virtual S7-1500 PLC*: Opens and closes the text boxes for creating the instance (virtual controller). Here we specify name, network addresses and type of the instance.

5. *Buttons*: Buttons for operating the selected instances.

⑥ *Instance list*: The list shows the available local instances. The instances can be resorted using the mouse cursor.

⑦ *LED displays*: The meaning of the LED is displayed when you move the mouse over it.

⑧ *Icons*:Icons for operating the instance

⑨ *Runtime Manager Port*: Here you open a port on the local PC.

⑩ *Virtual SIMATIC Memory Card*: Open an Explorer window here in which you select the path to the virtual memory card.

⑪ *Display messages*: Here you disable the PLCSIM Advanced messages in the Windows task bar for the duration of the operation.

⑫ *Function manual*: This is where you open the S7-PLCSIM Advanced Function Manual in a standard PDF viewer.

⑬ *Exit*: Exit logs off all instances and closes the Control Panel.

References

[1] Yongxin Liao et al. “Past, present and future of Industry 4.0 - a systematic literature review and research agenda proposal”. In: *International Journal of Production Research* 55 (Mar. 2017).

[2] Alexander Verl, Armin Lechler, and Jan Schlechtendahl. “Glocalized cyber physical production systems”. In: *Production Engineering* 6 (Dec. 2012).

[3] Laszlo Monostori et al. “Cyber-physical systems in manufacturing”. In: *CIRP Annals - Manufacturing Technology* 65 (Aug. 2016), pp. 621–641.

[4] Jay Lee, Behrad Bagheri, and Hung-An Kao. “A Cyber-Physical Systems architecture for Industry 4.0-based manufacturing systems”. In: *SME Manufacturing Letters* 3 (Dec. 2014).

[5] Ke-Sheng Wang. “Intelligent and integrated RFID (II-RFID) system for improving traceability in manufacturing”. In: *Advances in Manufacturing* 2 (June 2014), pp. 106–120.

[6] Z.X. Guo et al. “An RFID-based intelligent decision support system architecture for production monitoring and scheduling in a distributed manufacturing environment”. In: *International Journal of Production Economics* 159 (Jan. 2015), pp. 16–28.

[7] Hao Zhang et al. “A Digital Twin-Based Approach for Designing and Multi-Objective Optimization of Hollow Glass Production Line”. In: *IEEE Access* 5 (Oct. 2017), pp. 26901–26911.

[8] Duncan Mcfarlane et al. “The intelligent product in manufacturing control and management”. In: *J. EAIA* (Jan. 2002), pp. 54–64.

[9] Ricki Ingalls. “Introduction to simulation: introduction to simulation.” In: Jan. 2002, pp. 7–16.

[10] Pierre-Jean Erard and Pontien Déguénon. *SIMULATION PAR EVENEMENTS DISCRETS*. 1996.

[11] Flexsim Software Products. URL: https://www.flexsim.com/ [Access date: 04/2020].

[12] *FlexSim*. URL: https://en.wikipedia.org/wiki/FlexSim [Access date: 05/2020].

[13] Flexsim Software Products. *Flexsim User Manual*.

[14] *EMULATION*. URL: https://docs.flexsim.com/en/20.0/Reference/Tools/Emulation/ [Access date: 08/2020].

[15] Fabio Bellifemine, Giovanni Caire, and Dominic Greenwood. "Developing Multi-agent Systems with JADE". In: *Developing Multi-Agent Systems with JADE* (Feb. 2007), pp. 1–286.

[16] Fabio Bellifemine, Agostino Poggi, and Giovanni Rimassa. "Developing Multi-agent Systems with JADE". In: July 2000, pp. 89–103.

[17] Telecom Italia Lab. URL: http://jade.tilab.com/.

[18] Fabio Bellifemine, Giovanni Caire, and Dominic Greenwood. *Developing Multi-Agent Systems with JADE*. Wiley Series in Agent Technology. 2007.

[19] Ahmed R. Sadik and Bodo Urban. "A Holonic Control System Design for a Human & Industrial Robot Cooperative Workcell". In: May 2016, p. 119.

[20] *Lesson: All About Sockets*. URL: https://docs.oracle.com/javase/tutorial/networking/sockets/ [Access date: 06/2020].

[21] *Network Programming sing sockets*. URL: http://cs.gmu.edu/~setia/cs707/slides/sockets.pdf [Access date: 06/2020].

[22] *TCP/IP Protocol Services*. URL: http://diranieh.com/SOCKETS/TCPIP.htm [Access date: 06/2020].

[23] The OPC Foundation. *The Interoperability Standard for Industrial Automation & Other Related Domains*. URL: https://opcfoundation.org/ [Access date: 06/2020].

[24] *OLE for Process Control (OPC) Overview*. Emerson Process Management Group.

[25] Marcel Nicola et al. "SCADA Systems Architecture Based on OPC and Web Servers and Integration of Applications for Industrial Process Control". In: (Jan. 2018).

[26] Eugen Diaconescu and Cristian Spirleanu. "Communication Solution for Industrial Control Applications with Multi-agents Using OPC Servers". In: Oct. 2012.

[27] The OPC Foundation. URL: https://opcfoundation.org/ [Access date: 06/2020].

[28] Wolfgang Mahnke, Stefan-Helmut Leitner, and Matthias Damm. "OPC unified architecture". In: Mar. 2009, p. 7.

[29] *Why OPC UA Matters*. URL: https://www.ni.com/en-lb/innovations/white-papers/12/why-opc-ua-matters.html [Access date: 09/2020].

[30] B.M. Wilamowski and J.D. Irwin. *Industrial communication systems*. Apr. 2016, pp. 1–962.

[31] Mathworks. *What Is Hardware-In-The-Loop Simulation?* URL: https://www.mathworks.com/help/physmod/simscape/ug/what-is-hardware-in-the-loop-simulation.html [Access date: 05/2020].

[32] Bachir Mihoubi et al. “Hardware in the loop simulation for product driven control of a cyber-physical manufacturing system”. In: *Production Engineering* 14 (Mar. 2020).

[33] Frank D.Petruzella. *Programmable logic controllers*. 4th. 2011.

[34] W.Bolton. *Programmable logic controllers*. 5th. 2009.

[35] Siemens. *Simatic ET 200SP CPU 1512SP-1 PN manual*.

[36] *D. Ibrahim SD CARD PROJECTS USING THE PIC MICROCONTROLLER Newnes*. Jan. 2010.

[37] Components101. *Arduino Uno*. URL: https://components101.com/microcontrollers/arduino-uno [Access date: 09/2020].

[38] *Ethernet Shield*. URL: https://www.arduino.cc/Main/ArduinoEthernetShield [Access date: 08/2020].

[39] *Ethernet library*. URL: https://www.arduino.cc/en/reference/ethernet [Access date: 08/2020].

[40] V.Daniel Hunt. *RFID- A GUIDE TO RADIO FREQUENCY IDENTIFICATION*. 2007.

[41] *RC522 RFID Module*. URL: https://components101.com/wireless/rc522-rfid-module [Access date: 08/2020].

[42] *SPI library*. URL: https://www.arduino.cc/en/reference/SPI [Access date: 08/2020].

[43] Hind EL HAOUZI and André Thomas. “Design and validation of a product-driven control system based on a six sigma methodology and discrete event simulation”. In: *Production Planning and Control* 20 (Sept. 2009).

[44] Gerben Meyer, Gijs Roest, and N. Szirbik. “Intelligent Products for Monitoring and Control of Road-Based Logistics”. In: Sept. 2010, pp. 1–6.

[45] Duncan Mcfarlane et al. “Product intelligence in industrial control: Theory and practice”. In: *Annual Reviews in Control* 37 (Apr. 2013), pp. 69–88.

[46] *NetBeans IDE Java Quick Start Tutorial*. URL: https://netbeans.org/kb/docs/java/quickstart.html [Access date: 08/2020].

[47] Siemens. *SIMATIC TIA Portal STEP 7 Basic V10.5 Getting Started*. 2009.

[48] Siemens. *S7-PLCSIM Advanced Function Manual*. 2016.