

MASTER

Applications of MathWorks Tools in Model-Based Systems Engineering

Chatterjee, Joyeeta

Award date:
2020

[Link to publication](#)

Disclaimer

This document contains a student thesis (bachelor's or master's), as authored by a student at Eindhoven University of Technology. Student theses are made available in the TU/e repository upon obtaining the required degree. The grade received is not published on the document as presented in the repository. The required complexity or quality of research of student theses may vary by program, and the required minimum study period may vary in duration.

General rights

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain



Department of Mathematics & Computer Science
Model Driven Software Engineering W&I Section
Software Engineering & Technology Research Group



Applications of MathWorks Tools in Model-Based Systems Engineering

Masters Thesis

Joyeeta Chatterjee
(1307398)
Masters in Embedded Systems
Cyber-Physical Systems Track

Supervisors:

Dr. ir. Ion Barosan, Assistant Professor, Mathematics & Computer
Science, TU Eindhoven

Dr. Julien Schmaltz, Principal Consultant, ICT Netherlands B.V.

Eindhoven, November 2020

Abstract

In Industry 4.0, the concept of designing Cyber-Physical Systems (CPSs) is a rapidly emerging trend where computer-based algorithms monitor and control physical processes of the machinery. The increasing complexity in CPS poses a challenge for engineers who develop such systems. To solve this growing challenge, the aspects of Systems Engineering (SE) such as the Model-Based Systems Engineering (MBSE) approach, system architecture development and system analysis techniques can be used for technical decision-making involved in development of CPS. MathWorks offers a wide range of tools for modelling, analysis and simulation such as MATLAB, Simulink, Stateflow, and SimEvents. These tools have a great potential for designing complex CPSs. In this project, a research was conducted to employ a suitable combination of MathWorks tools in a CPS developed using MBSE methodology.

With the help of MathWorks tools, the supervisory-control of Fischertechnik Factory Simulation 24V was modelled, simulated and analyzed. To improve the system performance, bottlenecks in the original setup were identified and two types of modifications were introduced to the model. The first modification dealt with implementing parallel execution of independent actions in the model components. The second modification was focused on changing the system architecture by addition of another component in the model. In the modified model, the total execution time was reduced to half and the system throughput was doubled as compared to the original model. Lastly, the factory model developed with MathWorks tools was connected to the Digital Twin of Fischertechnik Factory Simulation 24V using MQTT messaging protocol. As a result, a live-link was established between the Simulink model and the Digital Twin model.

The combination of MathWorks tools - Simulink, Stateflow, SimEvents and MATLAB, considered in this project proved to be beneficial in improving the existing design of the supervisory-control of the Fischertechnik Factory Simulation 24V. Moreover, the establishment of live-link between Simulink model and Digital Twin paved a new pathway for conducting system validation of the Simulink implementations.

Contents

| | |
|--|------------|
| Contents | iii |
| List of Figures | vi |
| List of Tables | xi |
| 1 Introduction | 1 |
| 1.1 Problem context | 2 |
| 1.2 Project context and research questions | 3 |
| 1.3 System context | 5 |
| 1.4 Thesis outline | 5 |
| 2 Background and related work | 6 |
| 2.1 Overview of MBSE tools and MathWorks tools | 6 |
| 2.1.1 Integration of MBSE tools with MathWorks Tools | 7 |
| 2.1.2 MathWorks tools for CPS and MBSE | 9 |
| 2.2 MBSE testing environment at ICT | 12 |
| 2.3 Digital Twin technology | 16 |
| 2.3.1 Digital Twin application at ICT | 17 |
| <hr/> | |
| Methodologies and Applications of MathWorks Tools in MBSE | iii |

CONTENTS

| | | |
|----------|---|-----------|
| 2.3.2 | Communication to Digital Twin using MQTT | 18 |
| 3 | Methodology | 20 |
| 3.1 | Project requirements | 20 |
| 3.1.1 | Selection of suitable MathWorks tools | 20 |
| 3.1.2 | Assumptions in the functioning of factory model | 22 |
| 3.1.3 | Factors considered for system analysis and performance optimization | 25 |
| 3.2 | Step 1 - Designing benchmark model in Simulink | 25 |
| 3.3 | Step 2 - System analysis for identification of bottlenecks | 26 |
| 3.4 | Step 3 - Modifications for performance optimization | 27 |
| 3.4.1 | Parallel execution of independent actions | 27 |
| 3.4.2 | Addition of extra components in factory model | 27 |
| 3.5 | Step 4 - Integration of Simulink model with Digital Twin | 28 |
| 4 | Implementation and testing | 29 |
| 4.1 | Benchmark model in Simulink | 29 |
| 4.2 | Modified model with parallel execution of independent actions | 38 |
| 4.3 | Modified model with two rack feeders | 42 |
| 4.4 | Live-link between Simulink and Digital Twin | 47 |
| 5 | Conclusions and future work | 50 |
| | References | 52 |
| | Appendix | 55 |
| A | Architecture model in System Composer | 56 |

| | |
|---|-----------|
| B Benchmark model | 58 |
| B.1 Warehouse subsystem | 58 |
| B.2 Robot subsystem | 61 |
| B.3 Processing Station subsystem | 63 |
| B.4 Color Sorter subsystem | 65 |
| C Modified model for parallel execution of independent actions | 67 |
| C.1 Modifications in the Supervisory Control subsystem | 68 |
| C.2 Modifications in the Robot subsystem | 71 |
| D Modified model with two rack feeders | 73 |
| D.1 Modifications in the Supervisory Control subsystem | 73 |
| D.2 Modifications in the Warehouse subsystem | 76 |
| E MQTT in Simulink using MATLAB function block | 80 |
| E.1 Provisions in Warehouse subsystem | 80 |

List of Figures

| | | |
|------|--|----|
| 1.1 | Technologies that are transforming the manufacturing industry in Industry 4.0 [1]. | 1 |
| 1.2 | General architecture of a CPS. | 2 |
| 1.3 | Transition from document-centric to model-centric approach for information exchange in MBSE methodology. | 3 |
| 2.1 | Two ways to integrate IBM Rational Rhapsody with MathWorks Simulink. | 7 |
| 2.2 | Results from MathWorks Simulink model obtained within a SysML project in Cameo Systems Modeler [2]. | 8 |
| 2.3 | Mapping physical blocks in GENESYS (left) with Simulink elements (right) using the GENESYS Simulink Connector [3]. | 9 |
| 2.4 | A basic model in Simulink with two subsystems. | 10 |
| 2.5 | Simulink model with Stateflow chart and SimEvents blocks. | 11 |
| 2.6 | Two ways to connect system architectures in System Composer to implementations in Simulink. | 11 |
| 2.7 | Fischertechnik Factory Simulation 24V available at ICT Eindhoven (adapted from [4]). | 12 |
| 2.8 | Vacuum gripper robot [4]. | 13 |
| 2.9 | Barcodes on widget boxes corresponding to different widget colors [4]. | 14 |
| 2.10 | Automated warehouse [4]. | 14 |
| 2.11 | Multi processing station [4]. | 15 |

| | | |
|------|---|----|
| 2.12 | Sorting line with color detection [4]. | 16 |
| 2.13 | Interactivity between the physical and digital world in manufacturing process shown by Deloitte [5]. | 16 |
| 2.14 | Digital Twin of the Fischertechnik Factory Simulation 24V developed in Prespective software platform. | 17 |
| 2.15 | An illustration of data flow in MQTT network protocol. | 18 |
| 3.1 | Phases in the working of the factory model on the basis of widget movement. | 23 |
| 3.2 | Steps followed in the methodology to answer the research questions. | 25 |
| 4.1 | Simulink subsystems representing factory model components and supervisory control. | 29 |
| 4.2 | Inside the Supervisory Control subsystem in Simulink. | 30 |
| 4.3 | Information of input and output signals used in the Supervisory Control Stateflow chart (from Simulink Model Explorer). | 31 |
| 4.4 | Inside the Supervisory Control Stateflow chart. | 32 |
| 4.5 | Inside the Simulink function block in Warehouse subsystem which is used to calculate the system throughput. | 33 |
| 4.6 | An illustration of supervisory control implemented in the benchmark model for transfer of widgets from Warehouse to Color Sorter (Phase-1). | 34 |
| 4.7 | An illustration of supervisory control implemented in the benchmark model for transfer of widgets from Color Sorter to Warehouse (Phase-2). | 35 |
| 4.8 | Time graph of actions involved in benchmark model with only one widget in Phase-1 and Phase-2. | 36 |
| 4.9 | Graph for visualization of nine widgets moving through factory model components in Phase-1 and Phase-2 of the benchmark model. | 36 |
| 4.10 | Time Graph highlighting the waiting periods involved in benchmark model with two widgets in Phase-1 and Phase-2. | 37 |

LIST OF FIGURES

| | | |
|------|--|----|
| 4.11 | An illustration of supervisory control implemented in the modified model with parallel actions for transfer of widgets from Warehouse to Color Sorter (Phase-1). | 38 |
| 4.12 | An illustration of supervisory control implemented in the modified model with parallel actions for transfer of widgets from Color Sorter to Warehouse (Phase-2). | 39 |
| 4.13 | Time graph of actions involved in the modified model with parallel actions for only one widget in Phase-1 and Phase-2. | 40 |
| 4.14 | Graph for visualization of nine widgets moving through factory model components in Phase-1 and Phase-2 of the modified model with parallel actions. | 40 |
| 4.15 | Time Graph highlighting the waiting periods involved in the modified model with parallel actions for two widgets in Phase-1 and Phase-2. | 41 |
| 4.16 | Flow chart showing the decision making process involved in the modified Warehouse subsystem with two rack feeders. | 43 |
| 4.17 | Graph for visualization of nine widgets moving through factory model components in Phase-1 and Phase-2 of the modified model with two rack feeders. | 44 |
| 4.18 | Time Graph highlighting the waiting periods involved in the modified model with two rack feeders and two widgets moving in Phase-1 and Phase-2. | 45 |
| 4.19 | Graph of system throughput of the benchmark model of factory model measured in Simulink simulation run over a period of ten hours. | 45 |
| 4.20 | Inclusion of a MATLAB function block in the Warehouse subsystem of benchmark model | 48 |
| 4.21 | The rack feeder in the Digital Twin (subscriber) moved by sending MQTT messages from Simulink model (publisher). | 49 |
| A.1 | Components in the architecture model of the factory model in System Composer. | 56 |
| A.2 | Interfaces in the architecture model of the factory model in System Composer. | 57 |
| B.1 | Inside the Warehouse subsystem in Simulink. | 58 |
| B.2 | Information of input and output signals used in the Warehouse Stateflow chart (from Simulink Model Explorer). | 59 |

| | |
|--|----|
| B.3 Inside the Warehouse Stateflow chart. | 60 |
| B.4 Inside the Robot subsystem in Simulink. | 61 |
| B.5 Information of input and output signals used in the Robot Stateflow chart (from Simulink Model Explorer). | 61 |
| B.6 Inside the Robot Stateflow chart. | 62 |
| B.7 Inside the Processing Station subsystem in Simulink. | 63 |
| B.8 Information of input and output signals used in the Processing Station State- flow chart (from Simulink Model Explorer). | 63 |
| B.9 Inside the Processing Station Stateflow chart. | 64 |
| B.10 Inside the Color Sorter subsystem in Simulink. | 65 |
| B.11 Information of input and output signals used in the Color Sorter Stateflow chart (from Simulink Model Explorer). | 65 |
| B.12 Inside the Color Sorter Stateflow chart. | 66 |
| C.1 Inside the Supervisory Control subsystem modified for parallel actions. . . | 68 |
| C.2 Information of input and output signals used in the Supervisory Control Stateflow chart modified for parallel actions (from Simulink Model Explorer). . . | 69 |
| C.3 Inside the Supervisory Control Stateflow chart modified for parallel actions. . . | 70 |
| C.4 Inside the Robot subsystem modified for parallel actions. | 71 |
| C.5 Information of input and output signals used in the Robot Stateflow chart modified for parallel actions (from Simulink Model Explorer). | 71 |
| C.6 Inside the Robot Stateflow chart modified for parallel actions. | 72 |
| D.1 Inside the Supervisory Control subsystem modified for two rack feeders at Warehouse. | 73 |
| D.2 Information of input and output signals used in the Supervisory Control Stateflow chart modified for two rack feeders at Warehouse (from Simulink Model Explorer). | 74 |

LIST OF FIGURES

| | | |
|-----|---|----|
| D.3 | Inside the Supervisory Control Stateflow chart modified for two rack feeders at Warehouse. | 75 |
| D.4 | Inside the Warehouse subsystem (modified for two rack feeders). | 76 |
| D.5 | Information of input and output signals used in the Rack Feeder Stateflow charts (from Simulink Model Explorer). | 76 |
| D.6 | Information of input and output signals used in the Warehouse Stateflow chart modified for two rack feeders (from Simulink Model Explorer). | 77 |
| D.7 | Inside the rack feeder Stateflow chart. | 78 |
| D.8 | Inside the Warehouse Stateflow chart (modified for two rack feeders). | 79 |
| E.1 | Information of input and output signals used in the Warehouse Stateflow chart of the benchmark model with modifications for facilitating MQTT (from Simulink Model Explorer). | 80 |
| E.2 | Inside the Warehouse Stateflow chart of benchmark model (modified for facilitating MQTT). | 81 |

List of Tables

| | | |
|-----|--|----|
| 3.1 | Criteria for selection of MathWorks tools | 21 |
| 3.2 | Average timing values of actions involved in the factory model | 24 |
| 4.1 | Comparison of total execution time and average system throughput of the implementations of factory model in Simulink | 47 |

Chapter 1

Introduction

In recent years, several companies in the manufacturing industry have realized that they will soon reach the maximum limit of improvements possible in the physical and electronic components of machines. Therefore, the evolving high-tech industry around the world shifted its focus to the development of smarter software solutions in order to meet the increasing demand for advanced production machines, measuring and control systems and equipment. Over the past decade, a general trend towards digitalization and automation has been observed in the manufacturing or production industries. This ongoing transition is commonly referred to as the Fourth Industrial Revolution (Industry 4.0) [6]. It involves the application of information and communication technologies in the industries to enable networking between machines and processes. As shown in Figure 1.1, Industry 4.0 comprises of growth of technologies such as Internet of Things (IoT), automation, simulation and Big Data in the manufacturing industry [1].

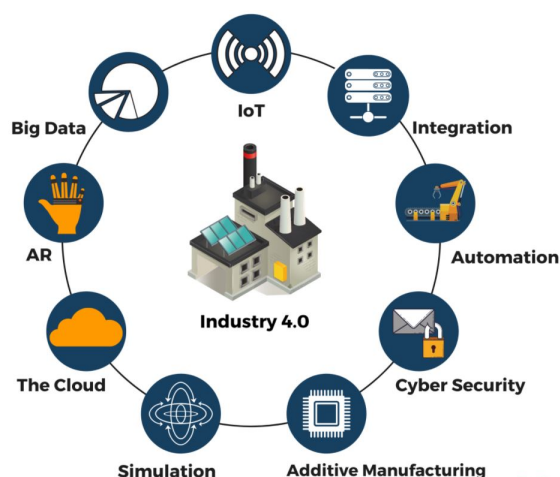


Figure 1.1: Technologies that are transforming the manufacturing industry in Industry 4.0 [1].

In Industry 4.0, the integration of physical processes with computation and networking has given rise to the concept of Cyber-Physical System (CPS) [7]. The general architecture of CPS is depicted in Figure 1.2 where physical mechanisms are monitored and controlled by computer-based algorithms through various communication networks. In combination with Internet of Things (IoT), CPSs have great potential for smart factories. CPSs have laid the foundation for many applications in the automotive, healthcare and manufacturing industries. In Europe, they play a major role in improving the economy and the quality of citizens' life [8]. Therefore, research and innovation in this field is highly encouraged so that the engineering techniques applied in designing CPS can be improved.

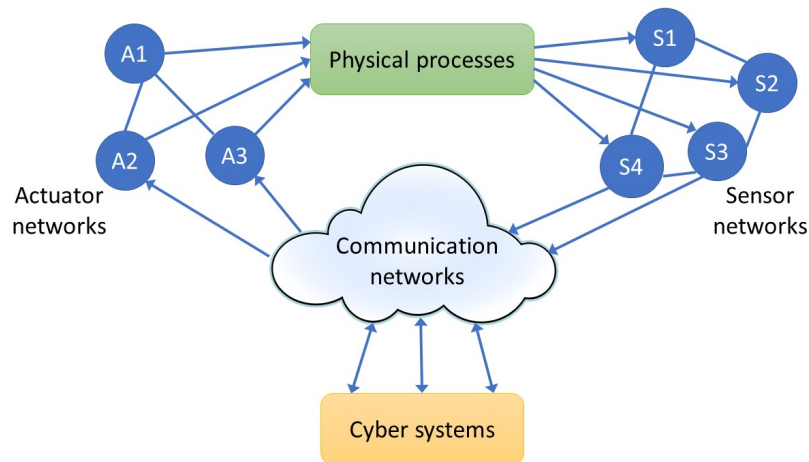


Figure 1.2: General architecture of a CPS.

The upcoming sections in this chapter include explanations of the main research problem handled in this project and the targeted research questions. Furthermore, a brief description is given for the approach followed in this project to answer the formulated research questions. In the end, a thesis outline is given for better readability purposes.

1.1 Problem context

The increasing complexity in CPS poses a challenge to the engineers who design and develop such systems. It has been observed that the strategies and concepts of Systems Engineering (SE) can be tremendously helpful in designing CPSs [9] [10]. The interdisciplinary approach of SE helps in management of complex systems throughout their life cycles. It considers both the technical and business needs of the customer and helps in critical decision-making for developing a quality product [11] (pp. 11-12). In SE, an architecture model is used to define the structure and behavior of the system. Generally, architecture

definition process is followed where several architecture alternatives are created and the most suitable architecture option is selected [11] (p. 64). Firstly, all the alternatives are assessed carefully. Afterwards, an architecture alternative is selected such that it meets all the system requirements and addresses stakeholder concerns in the best way possible. These quantitative assessments in SE are facilitated by a technique called System Analysis in which characteristics such as performance, cost, risks, feasibility and efficiency are analyzed for technical decision-making [11] (pp. 74-77).

Over the years, document-centric approaches proved to be increasingly inconvenient way of managing system requirements and maintaining consistency of complex systems. Thus, the methodology of Model-Based Systems Engineering (MBSE) evolved and engineers started using conceptual domain models as primary means of information exchange in system development [12]. This transition from document-centric to model-centric approach for information exchange in system development is depicted in Figure 1.3. MBSE helps in developing complex software with fewer bugs in significantly less amount of time and money. MBSE follows a top-down approach where the system development process starts from the highest-level abstractions and the functional requirements of the system are determined first. Then, the system is broken into a set of components and sub-components. The resulting implementation addresses the requirements directly.

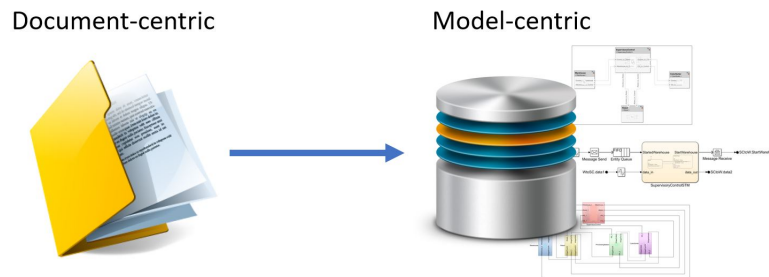


Figure 1.3: Transition from document-centric to model-centric approach for information exchange in MBSE methodology.

The MBSE methodology is a relatively new approach to software designing and is beneficial for the high-tech industry for the development of CPSs. Therefore, it is necessary to explore various tools suitable for MBSE domain in order to gain in-depth expertise and experience of the field.

1.2 Project context and research questions

In industrial projects and products, various software tools offered by MathWorks are widely used for model-based designing, simulation and analysis. Some of these tools have great

potential for designing complex CPSs. In this project, a research was conducted to employ a suitable combination of MathWorks tools in a CPS developed using MBSE practices. For this purpose, the Fischertechnik Factory Simulation 24V model was considered [4]. Moreover, a Digital Twin of the factory model available at ICT Eindhoven was used for further testing purposes.

For a system developed using MBSE, the research questions (RQs) formulated in this project are briefly described below.

RQ1: *How can the MathWorks tools help in optimizing the overall performance of a system?*

For developing efficient CPSs using MBSE methodology, it would be beneficial if the performance of the physical system could be determined and improved using smart software solutions. Hence, RQ1 aimed at exploring the possible ways in which MathWorks tools could be used to optimize the performance characteristics of a system such as throughput, latency and so on.

RQ2: *In the system development process, how can the MathWorks tools assist in parallel execution of actions?*

In large scale industries, it is desirable to execute independent industrial processes and machines in parallel so that the overall production rate could be increased. However, designing such parallel actions for components of large-scale intricate CPSs is a demanding process. Therefore, RQ2 aimed to investigate the role of system analysis and simulation with MathWorks tools in facilitating the parallel execution of actions in the system development process.

RQ3: *How can the MathWorks tools help in modification of system architecture required for improving the system performance?*

As described in section 1.1, architecture definition process of SE is useful in obtaining the best suitable architecture option for any given CPS. To create various architecture alternatives, the original architecture has to be studied in detail and the bottlenecks have to be identified. Thus, RQ3 aimed to inspect the use of MathWorks tools in recognition of limitations of a given system architecture and finding possible ways to modify it.

RQ4: *How to integrate the simulation of design models in MathWorks tools with testing on Digital Twin?*

For testing and validation purposes in CPSs, the emerging technology of Digital Twin is preferred over the real system as it reduces developmental risks and increases cost-effectiveness. The aspects of system design that are found to be infeasible after testing with the Digital Twin are discarded for the physical implementation. Therefore, it would

be advantageous if the real-time simulation of models designed using MathWorks tools could be integrated with testing on Digital Twin. Hence, RQ4 aimed to investigate the possible ways in which this crucial integration could be facilitated.

1.3 System context

After formulation of the research questions for this project, a research plan was defined to answer them. In the broader perspective, the plan focused on the use of MathWorks tools to improve an existing CPS developed using MBSE methodology. Therefore, the tasks listed below were expected to be accomplished.

- Use of MathWorks tools to develop a model of a system previously developed using MBSE methodology.
- System analysis and measurement of performance characteristics using selected parameters.
- Identification of bottlenecks in system functioning and architecture.
- Implementation of modifications to the original model in order to optimize the system performance.
- Real-time communication between models designed using MathWorks tools and the Digital Twin.

The details of the methods followed to complete the above-mentioned tasks and the related results are discussed later in this thesis.

1.4 Thesis outline

Following the introduction to the project, Chapter 2 consists of in-depth explanation of the background concepts and related work that are required for understanding the subsequent chapters of this thesis. Chapter 3 includes a description of the project requirements and assumptions as well as the steps followed to achieve the goals of this project. In Chapter 4, the resulting implementations are depicted and discussed in details. Lastly, Chapter 5 concludes the work done in this project and provides recommendations for future work.

Chapter 2

Background and related work

This chapter includes detailed description of the concepts and technology relevant to this graduation project, and the related work done in the same field. Firstly, different software tools suitable for MBSE applications are discussed, with a special focus on the tools offered by MathWorks. Then, the Fischertechnik Factory Simulation 24V available at ICT Eindhoven is described in great detail. Lastly, the innovative technology of Digital Twin is explained and related work carried out in this area is discussed.

2.1 Overview of MBSE tools and MathWorks tools

As the advantages of MBSE became popular over the years, several software tools were launched to provide solutions for modelling, designing and analyzing systems on the basis of MBSE practices. IBM Rational Rhapsody is one such tool which is widely used by system engineers and software developers [13]. It provides a visual system development environment that uses modelling languages such as Systems Modeling Language (SysML) and Unified Modeling Language (UML) to create graphical model designs. Moreover, this software possesses the capability to generate the code from models in programming languages such as C, C++, C# and Java.

Another popular software solution for MBSE is the Cameo Systems Modeler (formerly known as MagicDraw with SysML plug-in) [14]. It is a visual modelling environment that enables system engineers to create SysML models and diagrams in order to define various aspects of a system. The software facilitates engineering analysis for system requirements and designs that is required for the technical decision-making process.

The leading SE company, Vitech Corporation, has also developed two MBSE tools - CORE and GENESYS [15]. They cover all the domains of SE namely - requirements, behavior,

architecture, and verification and validation (V&V). In addition to engineering companies, many governmental organizations also use these tools as they support modelling with languages such as SysML, DoDAF (Department of Defense Architecture Framework) and so on.

2.1.1 Integration of MBSE tools with MathWorks Tools

Recently, various MBSE modelling environments have started featuring interoperability with the products of the computer software company, MathWorks. With these new features, integration is possible between MBSE tools and the widely known MathWorks products - MATLAB and Simulink. The former, MATLAB, is a numerical computing programming language as well as environment which uses matrix and array mathematics for analysis and design processes [16]. The latter, Simulink, is a graphical programming environment used for model-based designing, simulation and analysis of dynamical systems [17]. Simulink also features auto-generation of code from model in programming languages such as C and C++.

IBM Rational Rhapsody features integration of MathWorks Simulink models into Rational Rhapsody designs [18]. There are two ways to implement this integration as depicted in Figure 2.1. In the first kind of integration, a "black box" approach can be followed wherein the Simulink models can be represented by "Simulink blocks". Only the input/output ports of these blocks are seen in the UML or SysML model of Rational Rhapsody and are used to send or receive data from Simulink. During code generation, the Simulink-generated code is wrapped into the Rational Rhapsody-generated code. In the second kind of integration, a part of the system can be modelled using SysML in Rational Rhapsody and the generated code in C/C++ code can be used as an S-function in Simulink. S-functions are used to extend capabilities of Simulink such that blocks executing C/C++ code can be included in a model.

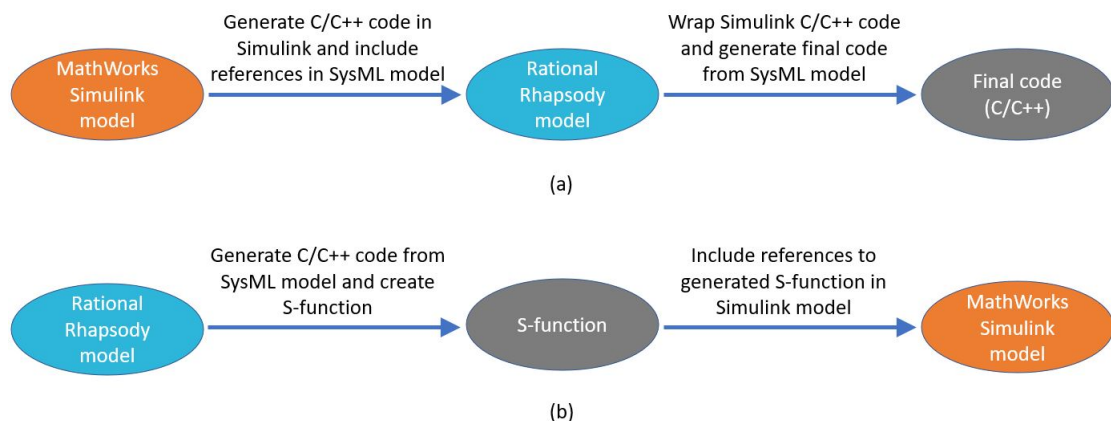


Figure 2.1: Two ways to integrate IBM Rational Rhapsody with MathWorks Simulink.

The integration between Rhapsody and Simulink depicted in Figure 2.1(a) was used to design a virtual prototype of flight control system for Unmanned Aerial Vehicle (UAV) [19]. The intended system was verified and validated early in the development cycle, and system specification errors were removed. For complex control systems, the integration of SysML modelling in Rational Rhapsody with simulation in Simulink is quite beneficial [20]. It allows engineers to follow SE practices in a SysML model while using Simulink to design control algorithms and plant behavior. Moreover, real-time simulations in Simulink also help in validation of system behavior.

In case of Cameo Systems Modeler, an integration with MATLAB is possible [21]. Using the Cameo Simulation Toolkit, a MATLAB/Simulink function can be directly called and the parameters can be passed from the UML/SysML model. After the MATLAB/Simulink model is executed, the related results can be seen back in the UML/SysML model as shown in Figure 2.2 [2]. In this way, the performance or physical properties of systems defined by the UML/SysML models can be evaluated by wide variety of analysis and simulations in MATLAB/Simulink.

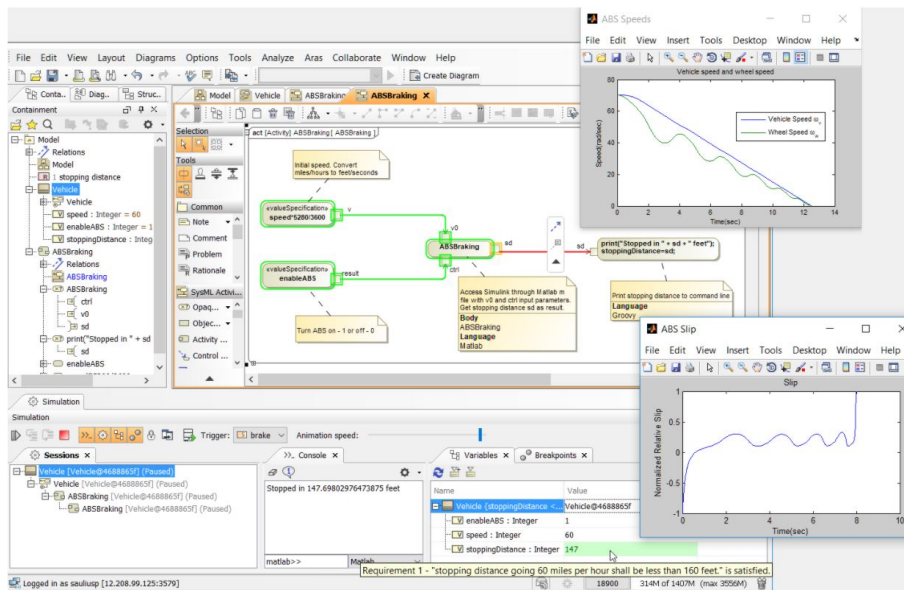


Figure 2.2: Results from MathWorks Simulink model obtained within a SysML project in Cameo Systems Modeler [2].

Vitech’s GENESYS also offers a dedicated Simulink connector which lets the user create the physical architecture of a Simulink model in GENESYS with the help of physical block diagram or flow Internal Block Diagram (IBD) as shown in Figure 2.3 [3]. Each component block in GENESYS can be mapped or associated with a Simulink element. Later, the GENESYS physical structure can be imported to Simulink for detailed design and analysis.

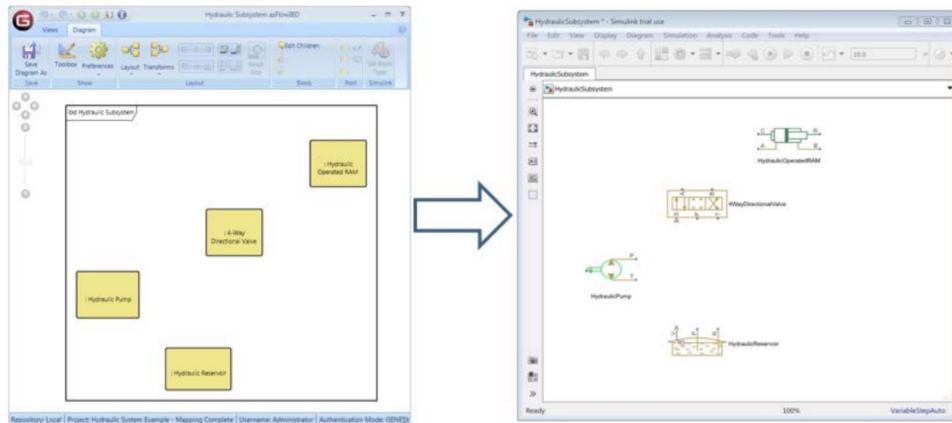


Figure 2.3: Mapping physical blocks in GENESYS (left) with Simulink elements (right) using the GENESYS Simulink Connector [3].

2.1.2 MathWorks tools for CPS and MBSE

Although SysML is widely used for MBSE, it is not suitable for simulation and analysis of dynamic systems. SysML is mainly useful for describing, testing and verifying of static architectural views and functional properties of the system. In contrast to SysML, Simulink is suitable for dynamic views of the system and also offers auto-generation of code from model which can be executed within other applications [22]. With the help of MATLAB algorithms and Simulink models, dynamic CPSs can be designed, simulated and analyzed. In addition, MathWorks also offers other supporting tools within MATLAB and Simulink for modelling complex CPS designs spanning multiple domains [23].

The MathWorks graphical control logic tool, Stateflow, enables development of state machine diagrams, flow charts, state transition tables, and truth tables in Simulink [24]. They can be used to model the reaction of MATLAB codes and Simulink models towards various input signals, events, and time-based conditions. Stateflow charts can communicate using data, events and messages. Here, messages are Stateflow objects used for communication of data locally or between Stateflow charts. The graphical animation of state transitions in Stateflow is useful for run-time debugging and analysis. Stateflow is useful for applications such as supervisory-control, task-scheduling and fault management.

The behavior of system performance can be modelled using another MathWorks tool called SimEvents [25]. It helps in analyzing event-driven systems and optimizing system performance on the basis of characteristics such as latency, throughput, and packet loss. SimEvents offers message-based communication and event-driven system modelling within the time-based simulation environment of Simulink. In SimEvents, an entity is a discrete item of interest which behaves the same way as a Stateflow message. In the Simulink library,

availability of SimEvents blocks such as Entity Generator, Queue, Terminator, Gate and Server facilitates modelling of queuing systems in Simulink. Moreover, the MATLAB Discrete Event System block helps in using a MATLAB algorithm in Simulink for customized discrete-event modelling.

To explain the working of Stateflow charts and SimEvents blocks, a basic Simulink model is shown in Figure 2.4. It has two separate subsystems for Stateflow chart (orange block) and SimEvents blocks (green block).

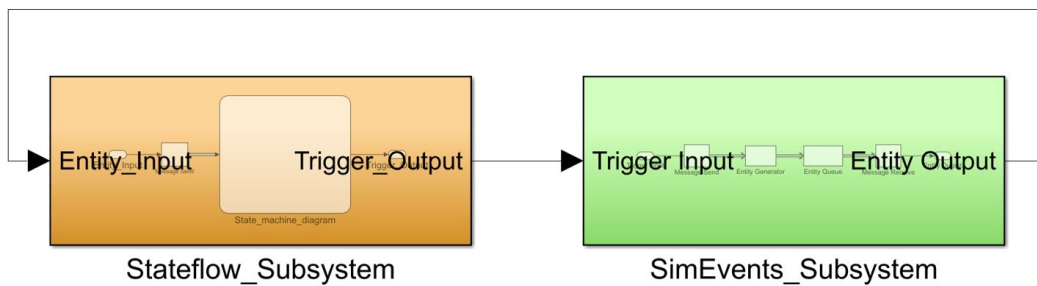


Figure 2.4: A basic model in Simulink with two subsystems.

The insides of the `Stateflow_Subsystem` (orange block in Figure 2.4) are depicted in Figure 2.5(a) and (b). It is a simple state machine which has two states `IDLE` and `START`. While in `IDLE` state, it waits for 2 seconds and then transitions to `START` state. While in `START` state, it sets the `Trigger_Output` equal to 1. This trigger is used as an input to the `SimEvents_Subsystem` (green block in Figure 2.4).

The insides of `SimEvents_Subsystem` (green block in Figure 2.4) are depicted in Figure 2.5(c). Firstly, a `Message Send` block is used to convert the Simulink input trigger signal to a discrete item of interest - a message that carries the original signal value. Due to this trigger, an entity is generated by the `Entity Generator` and sent to the `Entity Queue`. Then, the `Message Receive` block extracts signal value from the received message and writes it to the output signal port - `Entity_Output`. This signal is an input to the `Stateflow_Subsystem`.

Figure 2.5(a) shows that a `Message Send` block is used to convert the Simulink input signal to message before it enters the Stateflow chart. As shown in Figure 2.5(b), the state transition from `STATE` to `IDLE` occurs when there is a message available (i.e. `Entity_Input`). In the `IDLE` state, `Trigger_Output` is set to 0.

In this way, Stateflow charts and SimEvents blocks can be utilized within a Simulink model.

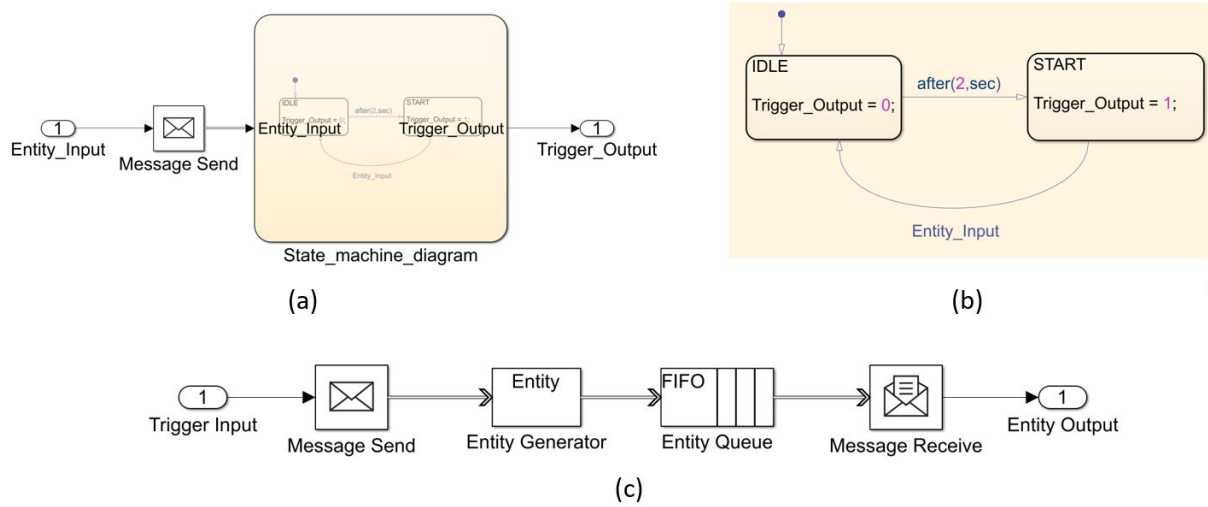


Figure 2.5: Simulink model with Stateflow chart and SimEvents blocks.

In 2019, MathWorks also launched a separate tool for MBSE and software design called System Composer. It can be used to create system architecture models in terms of components and interfaces [26]. These models could be analyzed by performing trade-off studies. System Composer can be linked to Simulink for modelling behavior, running simulations, generating C/C++ code for deployment to hardware. This bridged the gap between architecture models and implementation (design) models [27]. Figure 2.6 shows that the user can connect an architecture component in System Composer model to a Simulink model by either creating a link to an existing Simulink model or by auto-generating a new Simulink model.

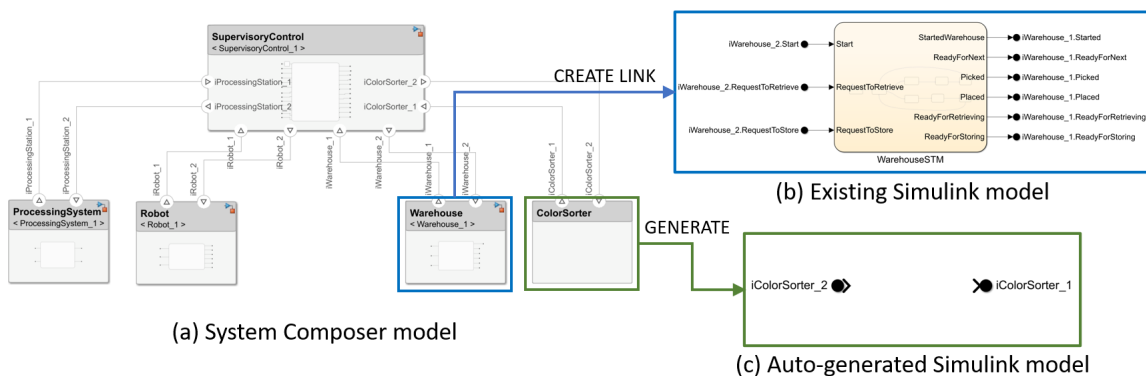


Figure 2.6: Two ways to connect system architectures in System Composer to implementations in Simulink.

2.2 MBSE testing environment at ICT

At ICT Eindhoven, the Fischertechnik Factory Simulation 24V model [4] is being used in numerous research based projects. This is a training model which helps in implementing and validating concepts on a small scale before extending to the industry level. As depicted in Figure 2.7, it is a combination of four components - Robot, Warehouse, Processing Station and Color Sorter. Nine small geometrically identical workpieces (three red, three white and three blue) are moved around these components to form a machining line. For the ease of understanding, the Fischertechnik Factory Simulation 24V will be simply referred to as the factory model and the colored workpieces will be referred to as widgets hereinafter.

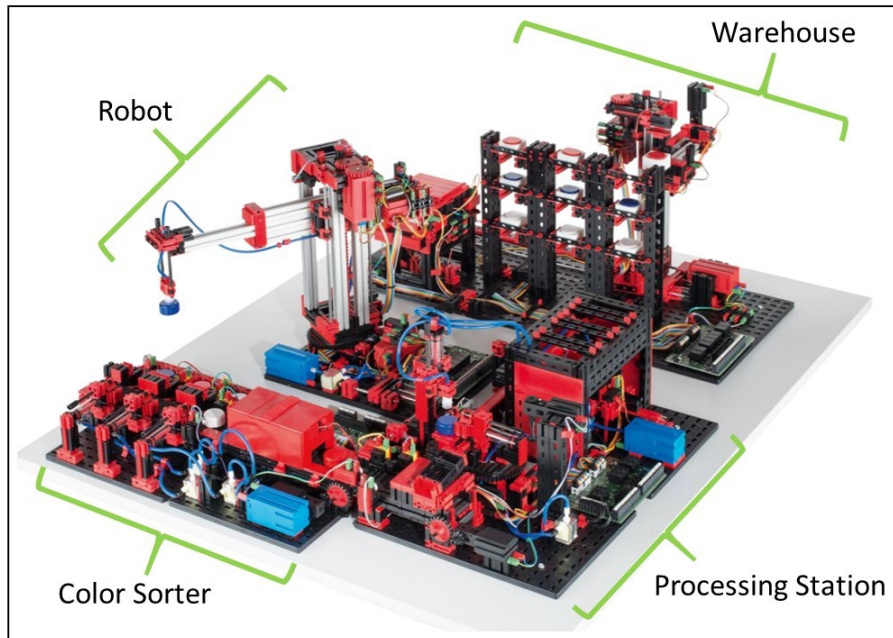


Figure 2.7: Fischertechnik Factory Simulation 24V available at ICT Eindhoven (adapted from [4]).

A widget is picked by the Robot from the Warehouse and placed at the Processing Station. Later, the Processing Station pushes the widget to the Color Sorter. In the end, the widget is picked up by the Robot at the Color Sorter and placed back at the Warehouse. The same process is followed for all the nine widgets.

ICT is also exploring the software engineering toolset, Dezyne, developed by Verum [28]. Thus, the Fischertechnik Factory Simulation 24V has been modelled on Dezyne and C++ code has been auto-generated. The setup uses five BeagleBone Black [29] boards for running the code of the factory model. Separate boards are connected to each of the four

factory model components. These boards directly communicate to the fifth board that is responsible for implementing supervisory-control. For this purpose, state machine diagrams are used and event-driven simulation is followed. In each of the four components, various state actions and transitions occur when the supervisory-control generates appropriate events for them. After a state action is completed or a state transition has occurred in a component, it generates an event to notify the supervisory-control so that events can be generated for the other components. In this way, the supervisory control supervises the working of the factory model components and the movement of the widgets through them.

The details of the individual components of the factory model and their working are as follows:

- **Robot** - It resembles an industrial robot and is used for picking up and placing the widgets from one location to another. It is capable of performing horizontal, vertical and rotational movements around a turntable. As shown in Figure 2.8, the Robot has a suction cup at the end of the horizontal arm. To pick up a widget, an airtight connection is made between the suction cup and the widget. The created vacuum helps the Robot to make a firm grip on the widget. After the Robot has moved to another location, it eliminates the vacuum and places the widget successfully. The Robot either picks up widgets at the Warehouse to place them at the Processing Station or picks up widgets at the Color Sorter to place them at the Warehouse.

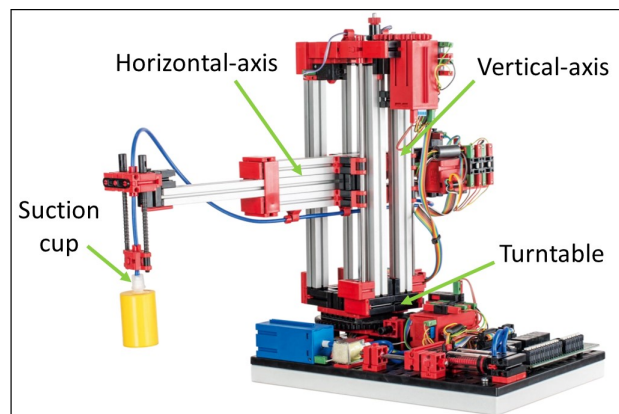


Figure 2.8: Vacuum gripper robot [4].

- **Warehouse** - It consists of a storage rack, a rack feeder and a conveyor system. The storage rack has nine boxes with barcodes on one side for carrying the widgets. There are three barcode patterns dedicated to the three widget colors - white, red, blue, as shown in Figure 2.9. The rack feeder carries widget boxes from the storage rack to the conveyor system and vice-versa. The conveyor system is responsible for moving the widget boxes from the rack feeder side to the Robot side and vice-versa.

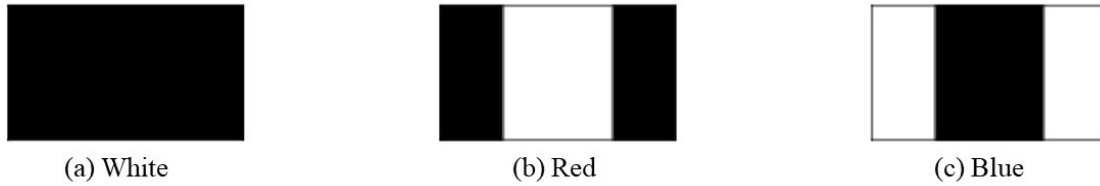


Figure 2.9: Barcodes on widget boxes corresponding to different widget colors [4].

The Warehouse and its important parts are depicted in Figure 2.10. There are two kinds of widget movements possible within the Warehouse. In the first kind of movement, the rack feeder retrieves a full widget box from the storage rack and places it on the belt of conveyor system. As the box reaches the other end of the conveyor belt, the Robot picks up the widget from the box. The rack feeder picks up the empty widget box from the conveyor belt and stores it back to the rack. In the second kind of movement, the rack feeder retrieves an empty widget box from the storage rack and places it on the belt of conveyor system. As the empty box reaches the other end of the conveyor belt, the Robot places the widget on the box. The rack feeder picks up the full widget box and stores it in the rack.

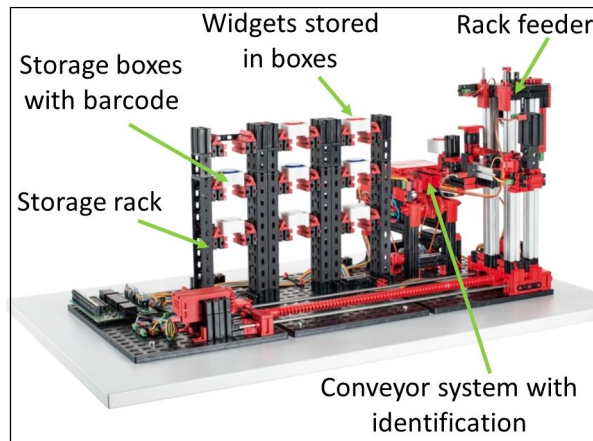


Figure 2.10: Automated warehouse [4].

There are 2 kinds of warehousing mechanisms possible in the factory model - static and dynamic. In static warehousing, for instance, each row or column in the storage rack can be assigned a widget color. As the Robot brings color sorted widgets from the Color Sorter, the rack feeder retrieves empty boxes from the Warehouse storage rack accordingly. For example, if the first row of storage rack at Warehouse is assigned white color and the Robot is about to place a white widget picked up from the Color Sorter, then, the rack feeder will retrieve an empty box from the first row of the Warehouse storage rack. On the other hand, in dynamic warehousing, there is no fixed assignment of widget color to any rack row or column. Instead an identification

mechanism is used by the conveyor system where a trail sensor tracks the light/dark differences in a barcode on the widget box. When the rack feeder places the widget boxes for the first time on the conveyor belt, their assigned color is detected using the barcode scheme shown in Figure 2.9. Later, when the Robot is about to place a color sorted widget on the conveyor belt, the rack feeder retrieves an empty widget box from the Warehouse storage rack corresponding to that widget color.

- **Processing Station** - It consists of several stations that simulate different processes on the widget as depicted in Figure 2.11.

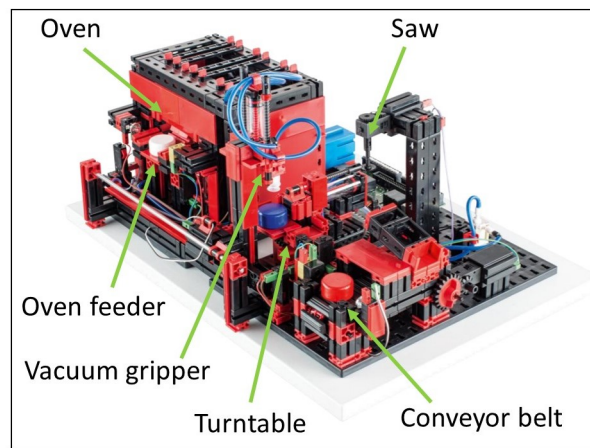


Figure 2.11: Multi processing station [4].

The processing starts at the oven when the Robot places the widget at the oven feeder. The oven feeder containing the widget is taken inside the oven. After the simulated firing process in the oven is over, a small vacuum gripper brings the widget to a turntable. At the turntable, some provisions have been made such that widget is positioned under a saw and made to wait there for a given time duration of processing. Lastly, the widget is pushed to the conveyor belt and sent to the Color Sorter.

- **Color Sorter** - It is used for automated separation of widgets on the basis of their color. The main components of Color Sorter are a conveyor belt, an optical color sensor inside a darkened sluice, three pneumatic cylinders and three widget storage locations as shown in Figure 2.12.

When a widget is placed on the conveyor belt by the Processing Station, it reaches the Color Sorter and goes through a darkened sluice which has an optical color sensor inside. After the widget color has been detected, it passes through a light barrier. Depending on the color value detected, the corresponding pneumatic cylinder is triggered with a delay after the light barrier has been halted by the widget. As a result, the widget is pushed into the one of the three chutes by a pneumatic cylinder and reaches a particular storage location assigned for that widget color. The storage

location closest to the color detection location has been assigned the color white, the center the color red and the furthest away the color blue. From these storage locations at the Color Sorter, the Robot picks up the sorted widgets for storage at the Warehouse.

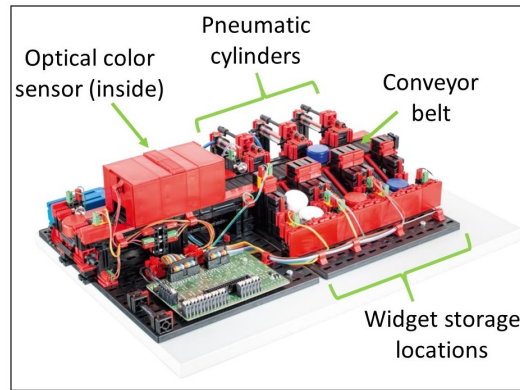


Figure 2.12: Sorting line with color detection [4].

2.3 Digital Twin technology

In the ongoing Industry 4.0 revolution, manufacturers have realized the significance of the virtual world in the industrial setup. The virtual prototypes of physical machines, termed as Digital Twins, help in validating a system design before it is physically implemented [30]. Using this cutting-edge technology, any industrial setup can be analyzed and tested virtually with a lower budget in lesser amount of time.

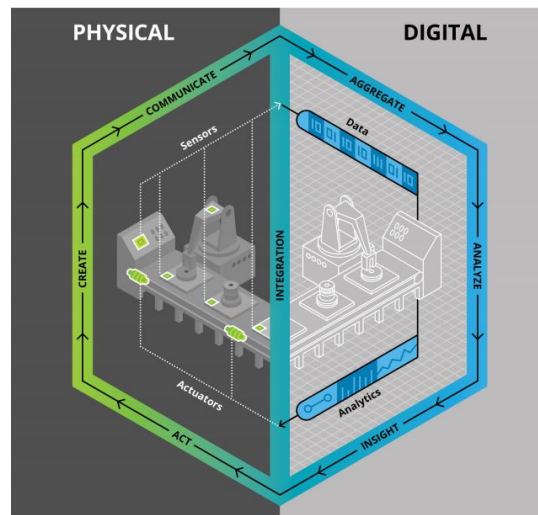


Figure 2.13: Interactivity between the physical and digital world in manufacturing process shown by Deloitte [5].

The physical world is integrated with the digital world as shown in Figure 2.13 [5]. The sensing and actuating occur in the physical world and then, the relevant data is sent to the digital world. This data is utilized to perform analysis and testing on the Digital Twin. The insight gained from this analysis helps in modifying the real system accordingly. As a result, system developers can detect potential risks and defects in a system at an early stage of development, thereby, increasing efficiency, reliability and cost-effectiveness.

2.3.1 Digital Twin application at ICT

The High-Tech Unit at ICT is exploring the applications of virtual world in industrial automation. For initial testing purposes, a Digital Twin of the Fischertechnik Factory Simulation 24 has been developed at ICT Eindhoven in collaboration with Prespective (formerly known as Unit040). The latter is a company that launched an interactive software platform of the same name for Digital Twin development [31]. Similar to the real system, the Digital Twin of the factory model is a combination of four individual components - Robot, Warehouse, Processing Station and Color Sorter, as shown in Figure 2.14. Currently, further improvements are underway such that continuous movement of virtual widgets can be illustrated in the Digital Twin similar to the movement of widgets in the real system.

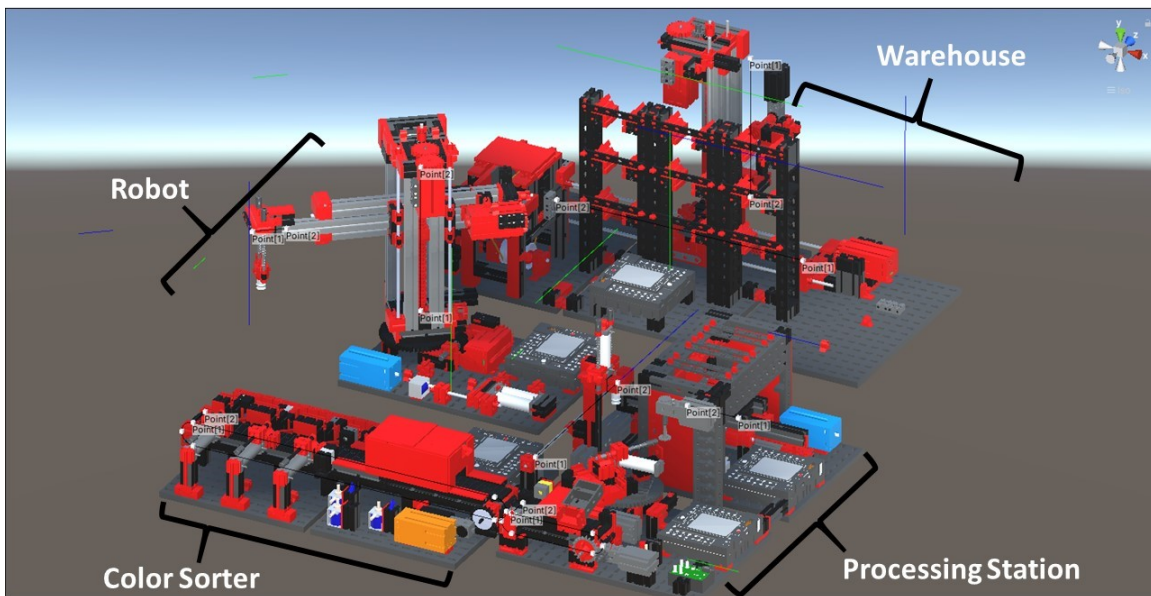


Figure 2.14: Digital Twin of the Fischertechnik Factory Simulation 24V developed in Prespective software platform.

2.3.2 Communication to Digital Twin using MQTT

Owing to its lightweight nature and high efficiency, Message Queuing Telemetry Transport (MQTT) is widely used in IoT devices. It allows bi-directional communication between multiple devices. As a messaging protocol, MQTT is considered to be highly scalable and reliable.

In MQTT, multiple clients connect to a broker which is a server responsible for receiving and routing messages [32]. The routing information of a message is contained in a topic. MQTT follows a publish-subscribe architecture style where a sender client publishes messages to a certain topic and the receiver client subscribes to the same topic to access the message. The broker matches the topics for each publisher and subscriber client, and delivers the messages accordingly. This process is demonstrated in Figure 2.15 where temperature sensor, laptop and smartphone are clients to the MQTT broker. The temperature sensor publishes a message - 22°C to the topic named "temp". The laptop and the smartphone subscribe to the same topic to receive the message.

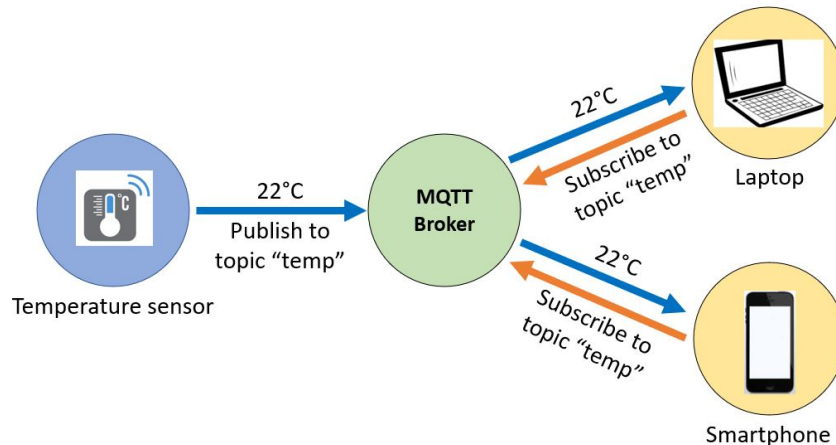


Figure 2.15: An illustration of data flow in MQTT network protocol.

Prespective has also developed a plug-in for communication to Digital Twins using MQTT messaging protocol. This plug-in was recently utilized in another TU/e graduation project to establish a live-link between a SysML model in IBM Rational Rhapsody and a Digital Twin developed in Unity3D [33]. Using an external broker and some additional C# code, data values sent from the model in Rational Rhapsody were successfully received at the Digital Twin in real-time. For future work, it was suggested that a research could be conducted to establish a similar connection between models in MATLAB/Simulink and Digital Twins using MQTT.

With the knowledge of above-mentioned background concepts and related work, a set of methods were specified to achieve the goals of this project. These methods are explained in the next chapter.

Chapter 3

Methodology

This chapter includes a description of the methodology that was followed to answer the research questions targeted in this project. Firstly, the project requirements and assumptions are specified. Then, the methods followed in this project are grouped into four major steps and discussed in details.

3.1 Project requirements

Before deciding the steps to be taken to answer the research questions, it was necessary to lay down the set of requirements for this project. These are listed as follows:

- Select a suitable combination of MathWorks tools.
- State the assumptions considered in the functioning of the factory model.
- Specify the factors to be considered for system analysis and performance optimization.

The above-mentioned requirements are thoroughly explained in the subsections below.

3.1.1 Selection of suitable MathWorks tools

From the wide range of available MathWorks tools, a few were selected and their combination was used for the design, simulation and analysis of the factory model. Table

3.1 specifies the criteria that was used to select the appropriate MathWorks tools for this project.

Table 3.1: Criteria for selection of MathWorks tools

| S.No. | Criteria |
|-------|---|
| 1 | Capability to perform model-based designing |
| 2 | Potential to model behavior using state machine diagrams |
| 3 | Ability to perform system analysis and performance optimization |
| 4 | Support for MQTT messaging protocol |

Since System Composer is an MBSE tool recently launched by MathWorks, it was the first tool that was considered in the tools selection procedure. An architecture model of the factory model was created with five components named as SupervisoryControl, Warehouse, Robot, ProcessingStation and ColorSorter. The interfaces were named as iWarehouse, iRobot, iProcessingStation and iColorSorter, and appropriate interface elements were defined for each one of them. The model is shown in Appendix A. Owing to its graphical environment for model-based designing, Simulink was considered for modelling the behavior of each component. This satisfied the first criteria mentioned in Table 3.1. Since the existing model of the factory model in Dezyne uses state machines for behavior modelling, it was decided that Stateflow charts would be used to implement the same in Simulink. This satisfied the second criteria mentioned in Table 3.1.

During the initial modelling process, it was realized that System Composer would not be useful for this project. System Composer is used for performing static analysis and trade-off studies using system parameters such as size, weight, power, cost and so on [34]. In case of the factory model, the electronic and mechanical properties of each individual component is fixed by the manufacturer. Thus, to improve the existing design of the factory model, the only parameter that could be analyzed and optimized was the time taken for each task. This would require timing values to be measured during run-time and analysis of a dynamic system. This could not be supported by System Composer. Moreover, it would not be possible to modify the lower-level system architecture of the factory model components as they are defined by the manufacturer. Hence, System Composer was deemed useless for this project.

After discarding the use of System Composer for this project, the modelling process was restarted with a Simulink model where each component of the factory model was represented by a subsystem. Inside each subsystem, state machines were included using Stateflow charts. The input and output ports of the Stateflow charts were connected to Simulink signals of the respective subsystems.

The third criteria mentioned in the Table 3.1 could be partially satisfied by logging Simulink signals and plotting their graphs against simulation time for system analysis. To study

and optimize the system performance in a better way, it was needed to introduce widgets in the simulation process. For visualization of widget movement in the factory model, it was decided that SimEvents blocks would be used in the Simulink models. As a result, the widgets were represented by SimEvents entities which are discrete items of interest. Since SimEvents entities essentially behave the same way as Stateflow messages, it was easy to model the widget behavior inside the state machines. In this way, a message-based communication could be modelled in Simulink and appropriate signals could be logged for timing measurements later.

Lastly, it was required to discover a method to implement MQTT messaging protocol in Simulink as mentioned by the fourth criteria in Table 3.1. The available Simulink blocks - MQTT Publish and MQTT Subscribe are only compatible with Raspberry Pi hardware [35]. Hence, these blocks could not be employed in this project. Since a toolbox for MQTT is available in MATLAB [36], it was decided that a suitable MATLAB code would be written to implement MQTT and the same would be included in the Simulink model using a MATLAB function block [37].

The final combination of MathWorks tools considered in this project is listed below:

- Simulink
- Stateflow
- SimEvents
- MATLAB

3.1.2 Assumptions in the functioning of factory model

Due to time constraints, it was not possible to fit all the functionalities of the factory model within the scope of this project. Therefore, the mechanism involved in the working of the factory model was simplified for implementation in this project. The assumptions defined for this purpose are stated as follows:

- **Happy path modelling:** All the Simulink models in this project were designed only for the happy flow of actions. Therefore, no exceptions or error conditions of the original factory model were considered.
- **Supervisory-control:** The Simulink models were designed only for the supervisory-control of the factory model. Hence, only the communication between the supervisory-control unit and the four factory model components was considered. The lower-level functionalities of the factory model components were not considered in this project.

- **Color of Widgets:** There was no distinction of the widgets on the basis of their color. It was assumed that all widgets were treated alike in the factory model and hence, the time taken for completing any action was the same for all widgets.
- **Default position of mobile components in the factory model:** There are three main mobile components in the factory model namely - rack feeder at the Warehouse, Robot, vacuum gripper at the Processing Station. In this project, it was assumed that the rack feeder was initially near the conveyor belt at Warehouse. Later, every time it stored a full or empty box, it rested near the storage rack and waited for the next command from Supervisory Control. The default position of the Robot was assumed to be located at the center of the factory model. Therefore, after placing a widget either at the Processing Station or the Warehouse, the Robot was assumed to have returned to the default position until the next pickup request arrived. Lastly, the default position of the vacuum gripper at the Processing station was assumed to be near the saw (hence, away from the oven feeder). This provided a clear pathway for the Robot to place a widget at the oven without colliding with the vacuum gripper at the Processing station.
- **Phases in factory model:** The working of factory model was divided into two phases which consisted of movement of the widgets through all the four components. Phase-1 consisted of the widget movement from the storage rack at Warehouse to the Processing Station and then, to the storage locations at the Color Sorter. Phase-2 consisted of widget movement from the storage locations at Color Sorter back to the storage rack at Warehouse. This is illustrated in Figure 3.1. In Phase-1, the widget movement from conveyor belt of Warehouse to the oven at Processing Station is assisted by the Robot whereas the widget movement from under the saw at Processing Station to the Color Sorter occurs with the help of a conveyor belt. In Phase-2, the widget movement from the storage locations at Color Sorter to the conveyor belt at Warehouse is also assisted by the Robot.

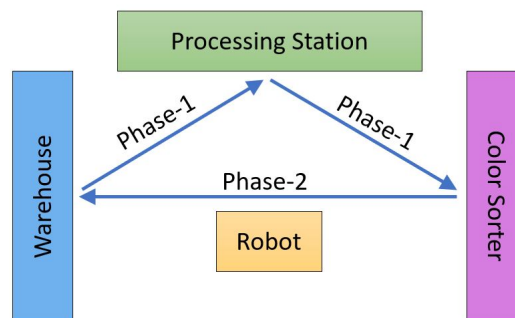


Figure 3.1: Phases in the working of the factory model on the basis of widget movement.

- **Timing values:** For simulating all actions of the factory model in Simulink, absolute-time temporal logic was used in Stateflow charts [38]. The timing values were meas-

ured in the physical model and an average value was used in the Stateflow temporal logic operator - **after**. Suppose an action took **n** seconds on an average to complete in the real factory model. In the corresponding state in the Stateflow chart, it was modelled as **after(n,sec)** such that the next action could only occur after **n** seconds have elapsed since the associated state became active. In the simulation process, this would mean that the particular factory model component was busy in completing the action for **n** seconds and could perform the next action only after **n** seconds have elapsed.

Table 3.2 shows the average time taken to perform all different kinds of actions in the physical model of the factory model.

- **Sequential and parallel execution of actions:** In sequential execution, it was assumed that the actions mentioned in Table 3.2 occur one after the other. On the other hand, in the parallel execution, it was assumed that the actions could occur at the same time in parallel with each other.

Table 3.2: Average timing values of actions involved in the factory model

| S.No. | Type of action | Average time taken to complete action (s) |
|-------|--|---|
| 1 | Initial startup of all components in factory model | 1 |
| 2 | Retrieval of first widget box by rack feeder resting near the conveyor belt at Warehouse | 13 |
| 3 | Retrieval of subsequent widget boxes by rack feeder resting near the storage rack at Warehouse | 8 |
| 4 | Storage of widget box by rack feeder at Warehouse | 8 |
| 5 | Pickup or placement of widget box on the conveyor belt by rack feeder at Warehouse | 2 |
| 6 | Movement of widget box to either sides of conveyor belt at Warehouse | 1 |
| 4 | Movement of Robot from default position to pickup location at Warehouse or Color Sorter | 7 |
| 5 | Pickup or placement of widget by Robot | 3 |
| 6 | Movement of Robot from pickup location at Warehouse or Color Sorter to placement location at Processing Station or Warehouse | 7 |
| 7 | Movement of Robot from placement location at Processing Station or Warehouse back to default position | 5 |
| 8 | Processing a widget in the oven at Processing Station | 13 |
| 9 | Processing a widget under the saw at Processing Station | 7 |
| 10 | Color detection and sorting of a widget | 7 |

3.1.3 Factors considered for system analysis and performance optimization

For carrying out system analysis and measuring the performance of the factory model in Simulink, two factors were considered. They are defined as follows:

- **Total execution time:** The total execution time of any Simulink model was given by the total time taken to complete both Phase-1 and Phase-2 for all the widgets present in the system i.e. the movement of widgets through all the four components of the factory model.
- **System throughput:** The Simulink models were designed in such a way that the movement of nine widgets through the factory model could be repeated. Therefore, the system throughput was given by the number of widgets moved through all the components of the factory model per unit time.

While analyzing an implementation of the factory model in Simulink, the total execution time and system throughput were calculated to determine the system performance. For optimizing the system performance, it was required that the total execution time was decreased and the system throughput was increased.

After the project requirements were set, a set of methods were followed to answer the targeted research questions. As shown in Figure 3.2, these methods are grouped into four major steps and explained in the subsequent sections of this chapter.

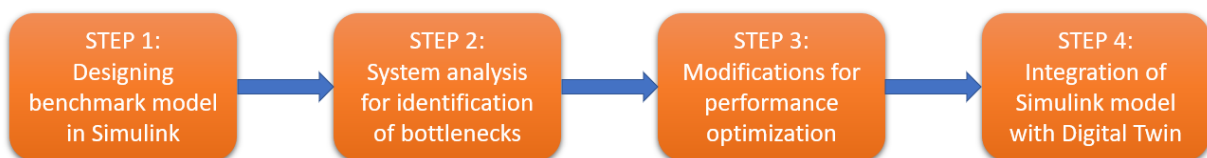


Figure 3.2: Steps followed in the methodology to answer the research questions.

3.2 Step 1 - Designing benchmark model in Simulink

Firstly, the behaviour of the supervisory-control of the original factory model setup was modelled in Simulink. Hereinafter, this model is referred to as the benchmark model of this project. It was expected that the total execution time would be maximum and the system throughput would be minimum in the benchmark model. Thus, the improved performance

of other modified implementations were supposed to be compared with the performance of this model.

To resemble the original factory model setup, a combination of sequential execution and parallel execution of actions was considered in the benchmark model. Except a few parallel actions, all other actions involved in Phase-1 and Phase-2 of the factory model occurred one after the other in a sequential manner. It was considered that the nine widgets were picked up by the Robot from the Warehouse and placed at the Processing Station one-by-one. Then, the widgets were moved from the Processing Station to the Color Sorter one after the other via the conveyor belt. After all the nine widgets were sorted at the Color Sorter, it was considered that they were picked up by the Robot and placed back at the Warehouse one-by-one.

In the Simulink model, five subsystems were created and named as Warehouse, Robot, Processing System, Color Sorter and Supervisory Control. For communication between these subsystems, Simulink buses and signals were used which contained set of signals of different data types. Inside each subsystem, Stateflow charts were used to model the behavior of factory model components and the supervisory-control using state machine diagrams. Additionally, the movement of widgets between the four components of the factory model was simulated using message-based communication in Stateflow and SimEvents blocks. This model and its working is explained in Chapter 4.

3.3 Step 2 - System analysis for identification of bottlenecks

For analyzing the benchmark model, the Simulink bus signals were logged for visualization in the Simulation Data Inspector [39]. During simulation, the signal values were changed by state actions and transitions in the Stateflow charts. Therefore, graphs were plotted with signal values in y-axis versus simulation time in x-axis. These graphs helped in measuring the total execution time of the benchmark model.

For calculation of system throughput, it was important to determine the total number of widgets which completed the movement through all the factory model components divided by the current simulation time. This could be measured in the Warehouse subsystem where widgets were finally stored in the storage rack at the end of Phase-2. For this purpose, a Simulink function block was added in the Warehouse subsystem [40]. It was configured for graphically defining a function - `CalculateThroughput()` whose output was the system throughput. The working of this Simulink function block is discussed in Chapter 4.

Using the graphs of signal values versus simulation time, the bottlenecks of the benchmark

model were identified. There were certain time periods where the factory model components stayed idle as they were waiting for commands from the supervisory control for a long time. Hence, the benchmark model was needed to be modified such that the wait periods in the factory model components could be either removed or utilized judiciously. As a result, the total execution time could be reduced and the system throughput could be increased, thereby, improving the system performance.

3.4 Step 3 - Modifications for performance optimization

To improve the system performance of the factory model, the benchmark model was modified in two ways. These are explained in the following subsections.

3.4.1 Parallel execution of independent actions

In the benchmark model, a combination of sequential and parallel execution of actions was considered. While carrying out the system analysis, it was observed that some actions in the original factory model setup were independent of each other and could occur at the same time. By executing independent actions in parallel with each other, the factory model components could be kept busy and the unnecessarily long wait periods could be utilized. To implement this, the Stateflow charts of the subsystems were modified accordingly. The modifications to the benchmark model and related graphs are thoroughly discussed in the next chapter.

3.4.2 Addition of extra components in factory model

In the graphs obtained from the modified model with parallel independent actions, it was observed that there were still some bottlenecks remaining in the system. There were certain time periods where a factory model component had completed its previous action but could not perform the next action because of its dependency on another factory model component which was busy completing its previous action.

It was realized that if there were multiple components or subcomponents of the same type working in parallel with each other, then, at least one of them will always be ready to perform the next action. Moreover, they can work with different widgets and accelerate the total execution process. For implementing this, the signal value versus simulation time graphs were carefully examined and the factory model component directly involved in the

bottleneck was identified. The Simulink subsystem of this factory model component was modified and extra subcomponents were added. This modification and the resulting graphs are explained in the next chapter.

3.5 Step 4 - Integration of Simulink model with Digital Twin

In this step, an aim was set to move the rack feeder in the Digital Twin by sending a value from the benchmark model during simulation in Simulink.

Using the MQTT in MATLAB toolbox, a MATLAB code was written for an MQTT publisher. To include this code in the Simulink model, a MATLAB function block was added in the Warehouse subsystem and a signal from the Stateflow chart was used as input to the block. Every time the value of the input signal changed, a new MQTT message containing the input signal value was published to a certain topic.

To facilitate the MQTT messaging protocol, the open source Eclipse Mosquitto broker was used [41]. With the help of engineers from Prespective, the Digital Twin was configured to act as an MQTT subscriber and was subscribed to same topic as the Simulink model. The value received in the MQTT message was used to rotate the DC motor of the rack feeder in the Digital Twin.

In this way, a live-link was established between the benchmark model in Simulink and the Digital Twin in Prespective. This working of this live link is explained in details in the next chapter.

Chapter 4

Implementation and testing

In this chapter, all the implementations of the factory model in Simulink are discussed. Their working mechanisms and signal value versus simulation time graphs are discussed in details. A comparison of all the models is drawn on the basis of the system performance i.e. total execution time and system throughput. Lastly, the working of the MQTT based live-link between the benchmark model in Simulink and the Digital Twin in Prespective is explained and related results are demonstrated.

4.1 Benchmark model in Simulink

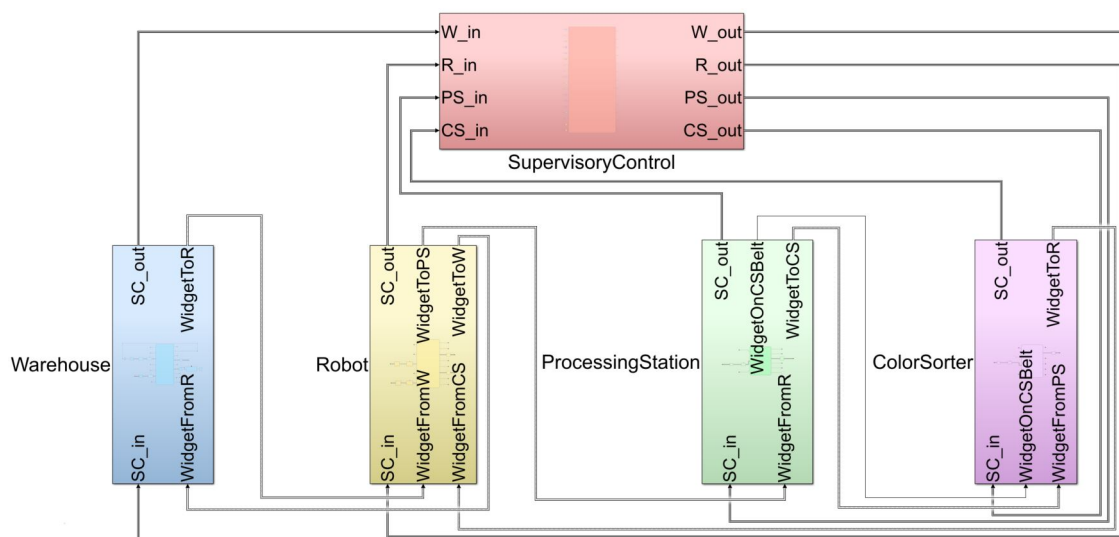


Figure 4.1: Simulink subsystems representing factory model components and supervisory control.

To resemble the working of the original factory model setup, five subsystems were considered in the benchmark model and were named as Supervisory Control (SC), Warehouse (W), Robot (R), ProcessingStation (PS) and ColorSorter (CS). As shown in Figure 4.1, the subsystems and their elements inside were color coded such that red represented Supervisory Control, blue represented Warehouse, yellow represented Robot, green represented Processing Station and purple represented Color Sorter. These color codes will be followed throughout this chapter hereinafter. Figure 4.1 also depicts connection of the Simulink bus signals between subsystems. These were used to communicate from the Supervisory Control to the four factory model components and vice-versa. Moreover, the four factory model components were also connected to each other using other Simulink buses for simulating the movement of widgets through them.

Inside these subsystems, Stateflow charts and SimEvents blocks were used to model the behavior. One such example is shown in Figure 4.2 where a Stateflow chart was used inside the Simulink subsystem of Supervisory Control. The input signals to this chart were the output of the four factory model components. Then, the output signals of the chart acted as input to the four factory model components. Figure 4.3 gives a detailed list of the input and output signals used for this Stateflow chart along with their data types and initial values.



Figure 4.2: Inside the Supervisory Control subsystem in Simulink.

| | Name | Scope | Port | Data Type | Initial Value |
|--|---------------------|--------|------|-----------|---------------|
| | StartedWarehouse | Input | 1 | double | |
| | StartedRobot | Input | 2 | double | |
| | StartedPS | Input | 3 | double | |
| | StartedCS | Input | 4 | double | |
| | WReadyToRetrieve | Input | 5 | double | |
| | PSReadyToReceive | Input | 6 | double | |
| | RPickedAtW | Input | 7 | double | |
| | RPlacedAtPS | Input | 8 | double | |
| | WReadyForNext | Input | 9 | double | |
| | RMoveCompleted | Input | 10 | double | |
| | PSProcessComplete | Input | 11 | double | |
| | CSWidgetAtLocation | Input | 12 | double | |
| | CSSortingComplete | Input | 13 | double | |
| | WReadyToStore | Input | 14 | double | |
| | CSReadyToRetrieve | Input | 15 | double | |
| | RPickedAtCS | Input | 16 | double | |
| | RPlacedAtW | Input | 17 | double | |
| | NextState | Local | | boolean | false |
| | Count1 | Local | | double | 0 |
| | Count2 | Local | | double | 0 |
| | TransferWToPS | Local | | boolean | false |
| | TransferCSToW | Local | | boolean | false |
| | x | Local | | double | 0 |
| | StartWarehouse | Output | 1 | double | 0 |
| | StartRobot | Output | 2 | double | 0 |
| | StartPS | Output | 3 | double | 0 |
| | StartCS | Output | 4 | double | 0 |
| | WRequestToRetrieve | Output | 5 | double | 0 |
| | RStartTransferWToPS | Output | 6 | double | 0 |
| | WPicked | Output | 7 | double | 0 |
| | PSPlaced | Output | 8 | double | 0 |
| | WRequestToStore | Output | 9 | double | 0 |
| | CSRequestToRetrieve | Output | 10 | double | 0 |
| | RStartTransferCSToW | Output | 11 | double | 0 |
| | CSPicked | Output | 12 | double | 0 |
| | WPlaced | Output | 13 | double | 0 |

Figure 4.3: Information of input and output signals used in the Supervisory Control State-flow chart (from Simulink Model Explorer).

The signals depicted in Figure 4.3 are used in the state actions and transitions shown in Figure 4.4. In this way, a state machine diagram of the Supervisory Control was modelled using Stateflow chart.

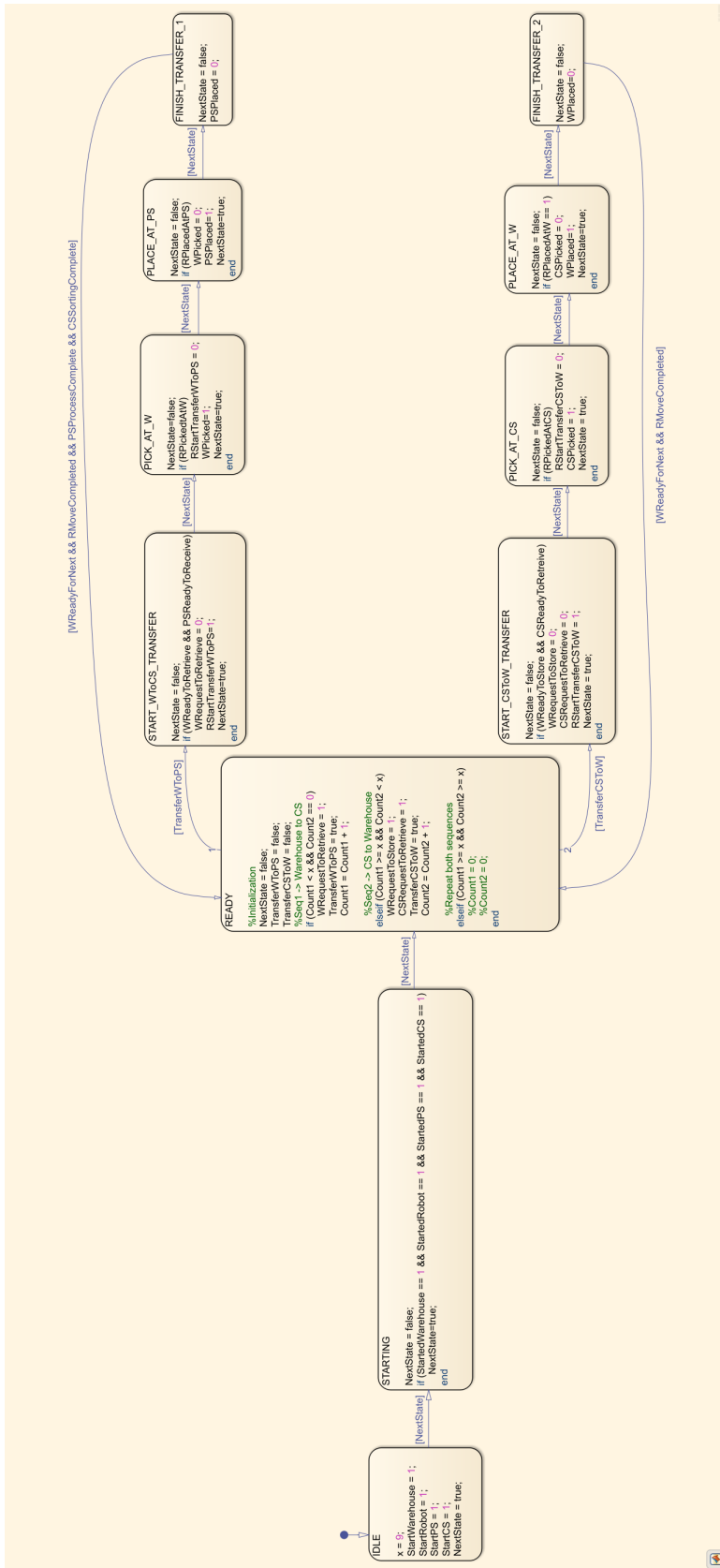


Figure 4.4: Inside the Supervisory Control Stateflow chart.

The state machines diagrams of Warehouse, Robot, Processing Station and Color Sorter were modelled in a similar manner. Moreover, these four subsystems also contained SimEvents blocks for facilitating the simulation of widgets movement through them. These Simulink models are shown in Appendix B.

With the help of the SimEvents blocks, widgets were represented by discrete items of interest called entities. The entities were configured as Simulink bus object with a signal called `WidgetNumber` that was used to differentiate between the nine widgets in the system. Every time an entity (widget) was generated in the Entity Generator SimEvents block, its attribute, `WidgetNumber`, was changed using the MATLAB code shown below. A pattern of repeating sequence was formed such that consecutively generated entities would have `WidgetNumber` value ranging from 1 to 9 which could be visualized later in time graphs.

```

1 % Pattern: Repeating Sequence
2 persistent SEQ;
3 persistent idx;
4 if isempty(SEQ)
5     SEQ = [1 2 3 4 5 6 7 8 9];
6     idx = 1;
7 end
8 if idx > numel(SEQ)
9     idx = 1;
10 end
11 entity.WidgetNumber = SEQ(idx);
12 idx = idx + 1;

```

In addition, the Warehouse subsystem consisted of a Simulink function block for calculation of system throughput at the location where the widgets were stored back in the rack at the end of Phase-2. As depicted in Figure 4.5, the total number of widgets entering the block as input were divided by the current simulation time to give the system throughput (in widgets/s) as output.

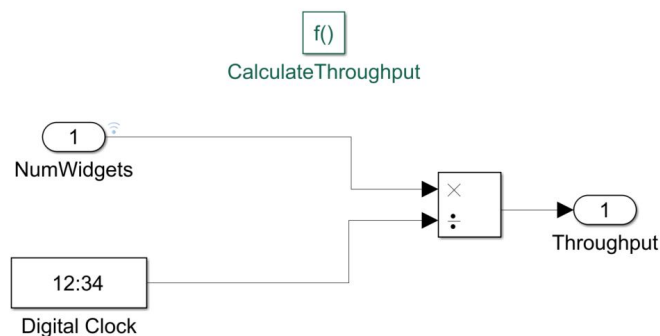


Figure 4.5: Inside the Simulink function block in Warehouse subsystem which is used to calculate the system throughput.

Figure 4.6 illustrates the working of the benchmark model in Phase-1 i.e., the movement of a widget from Warehouse to Color Sorter. Here, SC refers to the Supervisory Control, W refers to the Warehouse, R refers to the Robot, PS refers to the Processing Station and CS refers to the Color Sorter. In this diagram, the communication flow is shown such that the arrows going outward from SC blocks to other components blocks represent the commands sent from Supervisory control to the factory model components. Similarly, the arrows coming inward to the SC blocks represent the information sent by the factory model components to the Supervisory Control. The text inside the blocks represent the actions performed by the Supervisory Control and four factory model components in the benchmark model.

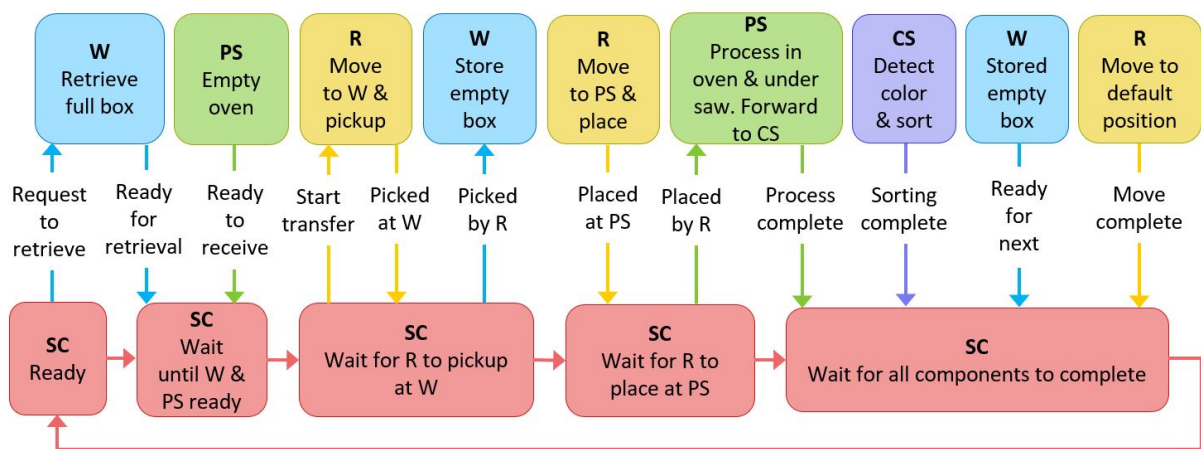


Figure 4.6: An illustration of supervisory control implemented in the benchmark model for transfer of widgets from Warehouse to Color Sorter (Phase-1).

Firstly, SC sends a request to W to retrieve a full box of widget. W retrieves the box and notifies SC that it is ready. Since there is no widget being processed at the PS oven, it is automatically ready to receive a widget. Hence, it notifies SC about the same. As both W and PS are ready, SC directs R to start the transfer of widget from W to PS. R moves to W, picks up the widget and informs SC. Then, SC informs W about the pickup done by robot so that W can store the empty box back in the rack. R moves with the widget to PS and places it at the oven. As SC is notified by R about the widget placement, it informs PS so that the processing can begin in the oven and be continued under the saw. Lastly, SC waits for all the factory model components to complete their ongoing processes. As the process is complete in PS, it forwards the widget to CS and informs SC about the same. CS detects the widget color, sorts it accordingly and informs SC. By this time, W has stored the empty box back in the rack and R has moved to its default position. When all the components have completed their actions, SC goes back to its ready state.

Figure 4.7 illustrates the working of the benchmark model in Phase-2 i.e., the movement of a sorted widget from Color Sorter back to the Warehouse. At the end of Phase-1 SC

comes back to its ready state and requests W to retrieve an empty box. At the same time, it also requests CS if it is ready for retrieval of widget by Robot. When W is ready with an empty box and CS is ready with a sorted widget, SC directs R to start the transfer of sorted widget from CS back to W. R moves to CS and picks up the sorted widget, then moves to W and places the widget in the empty box. As SC is informed about this placement, it informs W about the same so that it can store the full box in the rack. Lastly, SC waits for W to complete the storage process and R to move to its default position. When W is ready for next process to begin and R has completed its move, SC goes back to its ready state.

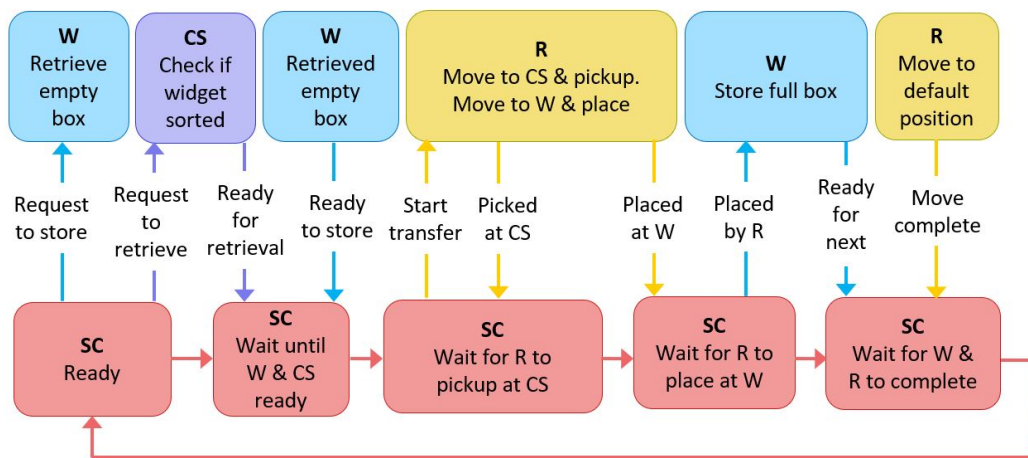


Figure 4.7: An illustration of supervisory control implemented in the benchmark model for transfer of widgets from Color Sorter to Warehouse (Phase-2).

The process explained in Figure 4.6 and 4.7 is for a single widget present in the system. During simulation of such a process in the benchmark model, graphs were obtained for Simulink signal values versus simulation time. This is shown in Figure 4.8. The dotted lines with a circle at the top represent the widget at various factory model components and continuous lines represent input/output signals in the subsystems of the benchmark model. The whole simulation process has been broken down into Phase-1 and Phase-2 wherein the individual actions of Warehouse (W), Robot (R), Processing Station (PS) and Color Sorter (CS) and their communication to the Supervisory Control (SC) are depicted in the graph.

In the benchmark model simulation, the same process was repeated for all nine widgets such that Phase-1 of all the nine widgets was completed consecutively first. When all the widgets were sorted by the Color Sorter, Phase-2 began and all the widgets were transferred back to the Warehouse. This movement of widgets is visualized with the help of the graph shown in Figure 4.9.

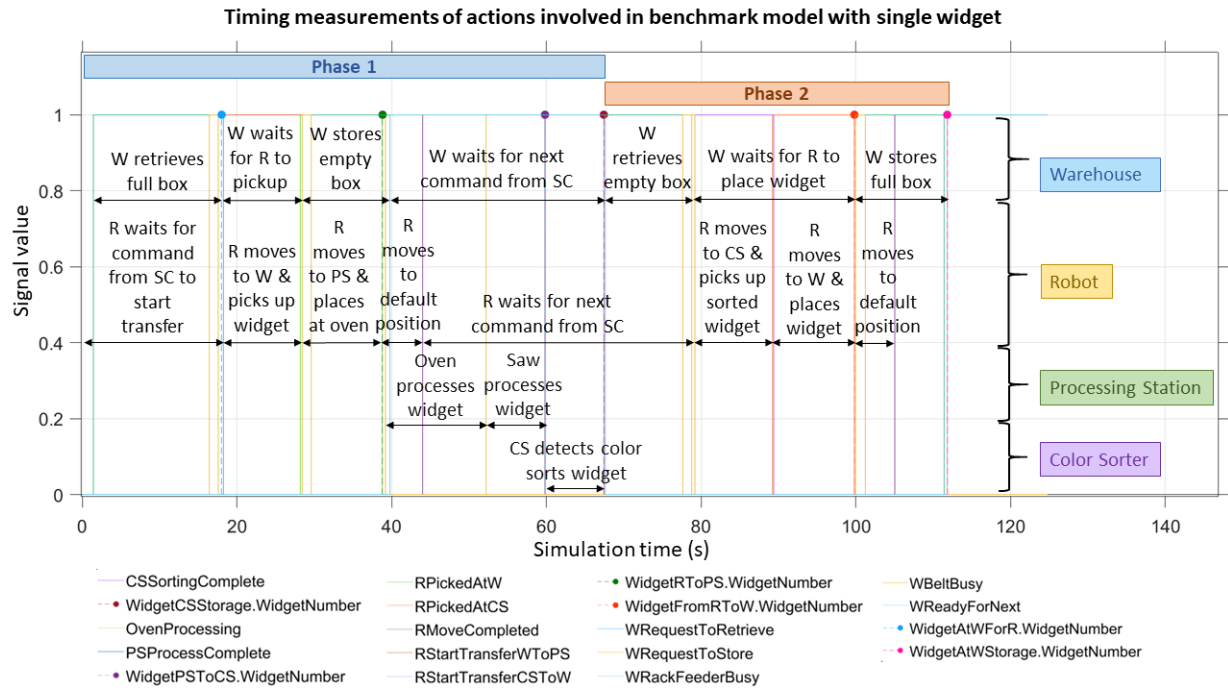


Figure 4.8: Time graph of actions involved in benchmark model with only one widget in Phase-1 and Phase-2.

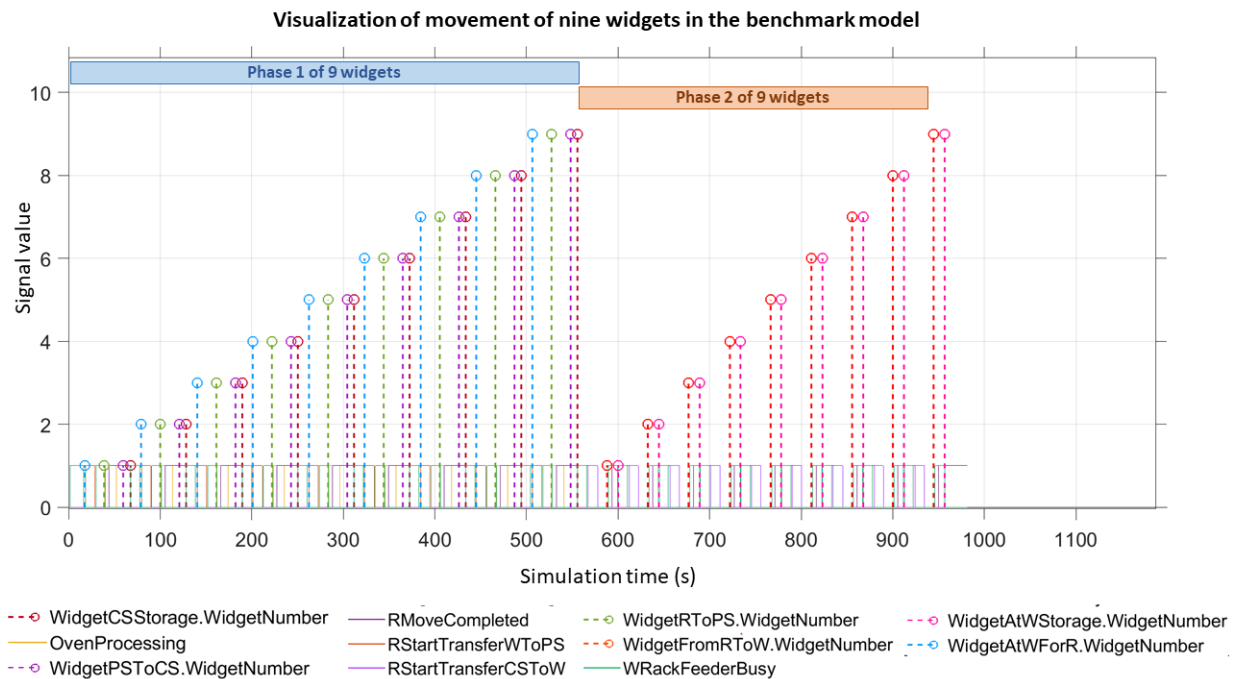


Figure 4.9: Graph for visualization of nine widgets moving through factory model components in Phase-1 and Phase-2 of the benchmark model.

In Figure 4.9, the nine widgets can be distinctively recognized by the signal values ranging from 1 to 9. The four dotted lines with circle top represent each widget in Phase-1 to be located at Warehouse conveyor belt (blue), oven at Processing Station (green), conveyor belt between Processing Station and Color Sorter (violet) and storage location at Color Sorter (maroon). In Phase-2, widgets are represented to be located at Warehouse conveyor belt (red) and Warehouse storage rack (pink).

To look closely into the working of the benchmark model for multiple widgets, the graph of signal value versus simulation time was plotted for two widgets moving in Phase-1 and Phase-2. It was realized there were certain time periods where the factory model components - Warehouse and Robot wait for the commands from Supervisory-Control for a long time. Instead of performing the next action, they stay idle. These waiting periods have been highlighted in Figure 4.10.

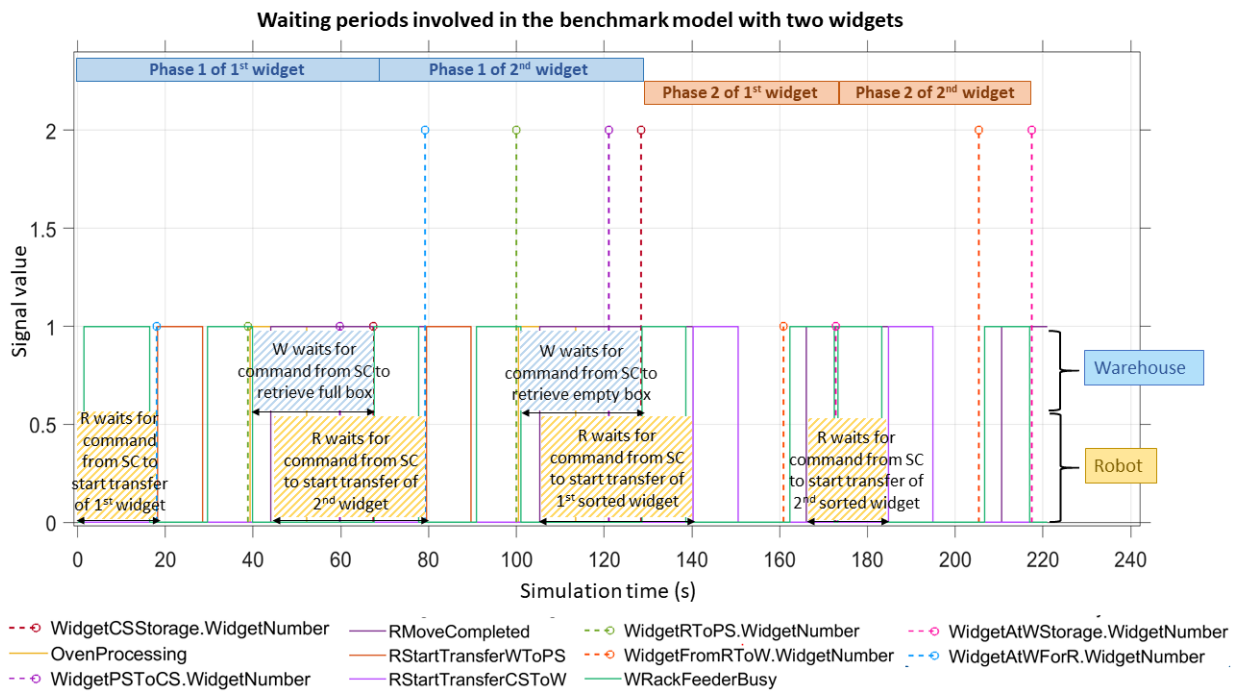


Figure 4.10: Time Graph highlighting the waiting periods involved in benchmark model with two widgets in Phase-1 and Phase-2.

These waiting periods and the actions succeeding them were studied closely. It was observed that there are certain actions that could be performed within these waiting periods. These independent actions could be performed in parallel with each other and the processes involved in factory model components Warehouse and Robot could be completed faster. Hence, the Stateflow charts were modified for enabling the parallel execution of these independent actions and are discussed in the next section.

4.2 Modified model with parallel execution of independent actions

The modified Simulink subsystems and Stateflow charts of Supervisory Control and Robot are depicted in Appendix C. The Warehouse subsystem itself was not modified but the actions related to it were modified in the Supervisory Control subsystem.

Figure 4.11 explains the working of this modified model in Phase-1. Firstly, SC sends a request to W retrieve a full box. Unlike the benchmark model, SC also directs R to start transfer of widgets from W to PS. Therefore, R moves to W and waits there until W retrieves a full box. As W is ready with a full box, it informs SC which, in turn, directs R to pickup the widget from the box at W. After R picks up the widget and informs SC, SC notifies W about the same so that it can store the empty box. In the meantime, R moves to PS. Since there is no widget being processed at the PS oven, it is automatically ready to receive a widget. Hence, it notifies SC about the same. SC directs R to place the widget as soon as it reaches PS. When R successfully places the widget, it informs SC. SC informs PS about the widget placement so that processing in the oven can be started and be continued under the saw. Without waiting for the components to finish their actions, SC returns to ready state. The independent actions of W storing the empty box, R moving to its default position, PS processing the widget and CS sorting the widget are completed in parallel without SC waiting for them.

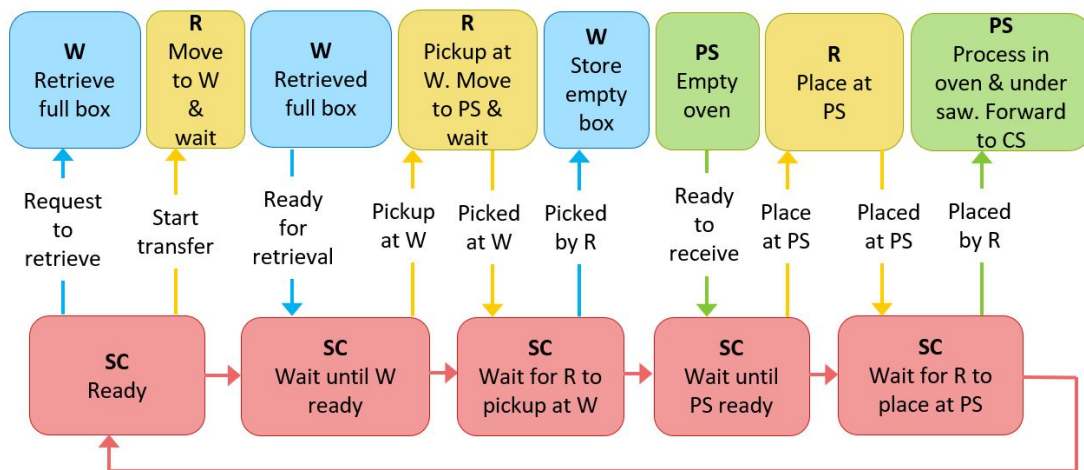


Figure 4.11: An illustration of supervisory control implemented in the modified model with parallel actions for transfer of widgets from Warehouse to Color Sorter (Phase-1).

Figure 4.12 explains the working of the modified model in Phase-2. As SC is back to its ready state at the end of Phase-1, it starts Phase-2 and sends a request to W to store a sorted widget. At the same time, it directs R to move to CS and wait until a sorted widget

is ready for pickup. SC then requests CS for retrieval of a sorted widget. When CS informs SC that a sorted widget is ready, SC directs R to pickup up the widget at CS. R informs SC about the pickup at CS. Meanwhile, W completes its action of retrieving an empty box and informs SC. SC directs R to place the widget at W after it reaches W. R places the widget at the empty box at W and informs SC. SC passes this information to W so that it can start storing the full box back to the rack. Without waiting for the components to finish their actions, SC returns to ready state. The independent actions of W storing the full box and R moving to its default position are completed in parallel without SC waiting for them.

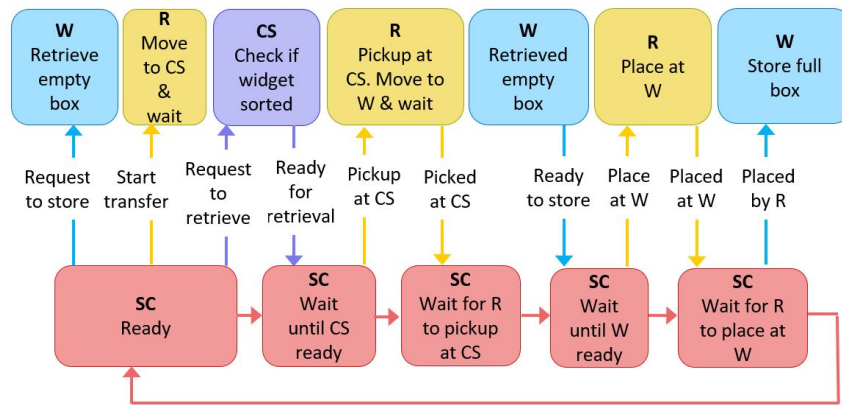


Figure 4.12: An illustration of supervisory control implemented in the modified model with parallel actions for transfer of widgets from Color Sorter to Warehouse (Phase-2).

Figure 4.13 depicts the graph obtained for the modified model with parallel execution of independent actions containing only one widget in the system. It shows the movement of widget in Phase-1 and Phase-2 wherein the individual actions of Warehouse (W), Robot (R), Processing Station (PS) and Color Sorter (CS) and their communication to the Supervisory Control (SC) are depicted. The dotted lines with a circle at the top represent the widget at various factory model components and continuous lines represent input/output signals in the subsystems of the modified model. It is important to notice that Phase-1 and Phase-2 overlap in this modified model as opposed to them occurring one after the other in the benchmark model.

The simulation process was repeated for all nine widgets in the modified model such that Phase-1 of all the nine widgets was completed consecutively followed by Phase-2 of all the widgets. This movement of widgets is visualized with the help of the graph shown in Figure 4.14. It is important to notice that the end of Phase-1 of ninth widget and beginning of Phase-2 of first sorted widget overlapped in the modified model. Moreover, the dotted lines for widgets at different locations in the factory model in Phase-1 overlap each other and are not plotted in a sequential manner as they were in the benchmark model. This showed that the independent actions in the individual factory model components were indeed performed in parallel.

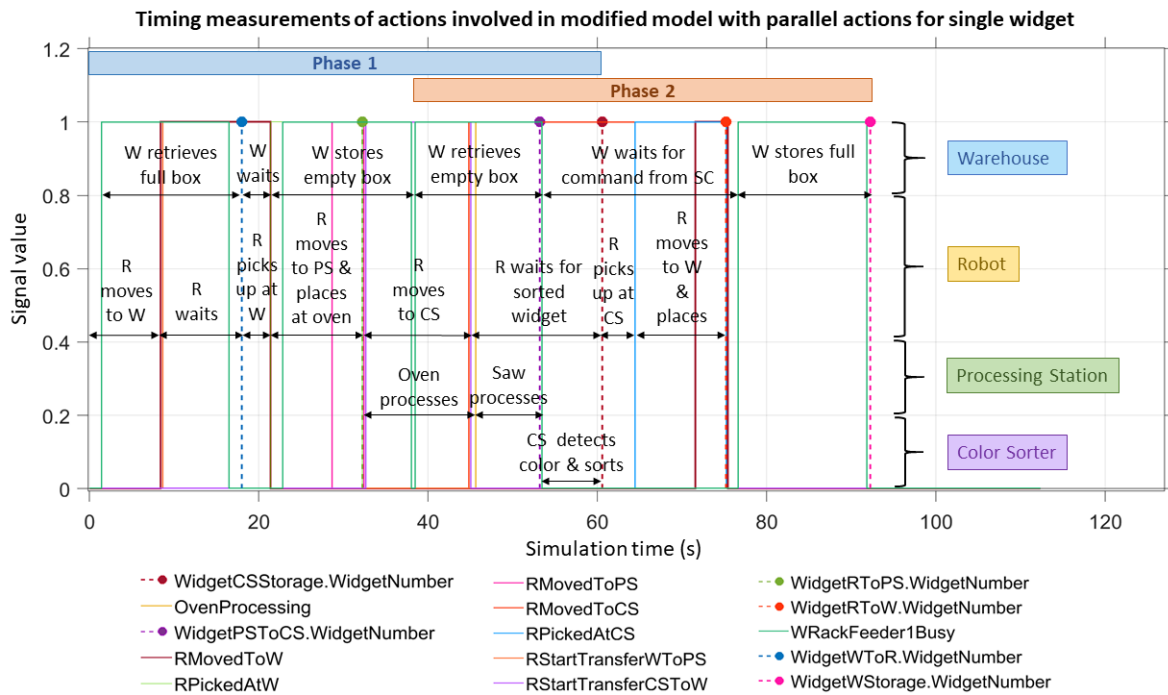


Figure 4.13: Time graph of actions involved in the modified model with parallel actions for only one widget in Phase-1 and Phase-2.

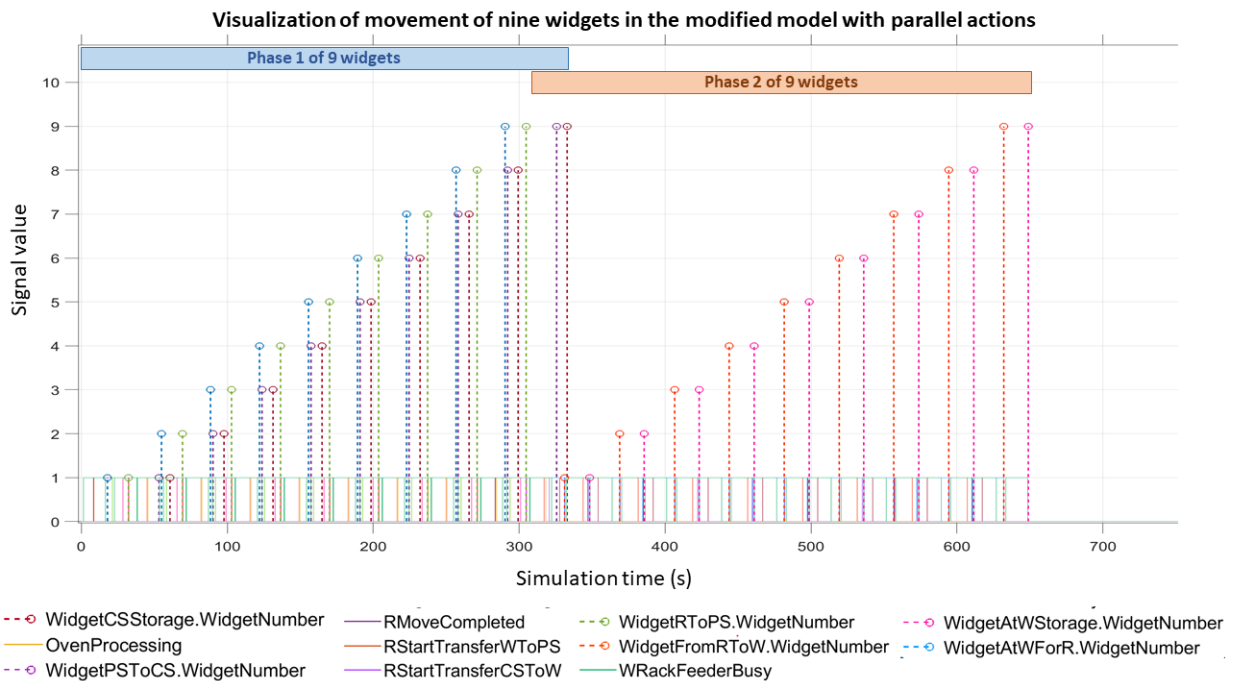


Figure 4.14: Graph for visualization of nine widgets moving through factory model components in Phase-1 and Phase-2 of the modified model with parallel actions.

To examine the improvements in the performance of the modified model as compared to the benchmark model, a graph of signal value versus simulation time was plotted for two widgets moving in Phase-1 and Phase-2 as shown in Figure 4.15. It was noticed that both Phase-1 and Phase-2 of the two widgets overlapped with each other. Moreover, the graph showed that the most of the waiting periods of benchmark model were utilized to perform independent actions in a parallel manner in the modified model. Hence, Mathworks tools assisted in performing parallel execution of actions in the factory model. This implementation and its results answered the second research question formulated in this project (RQ2).

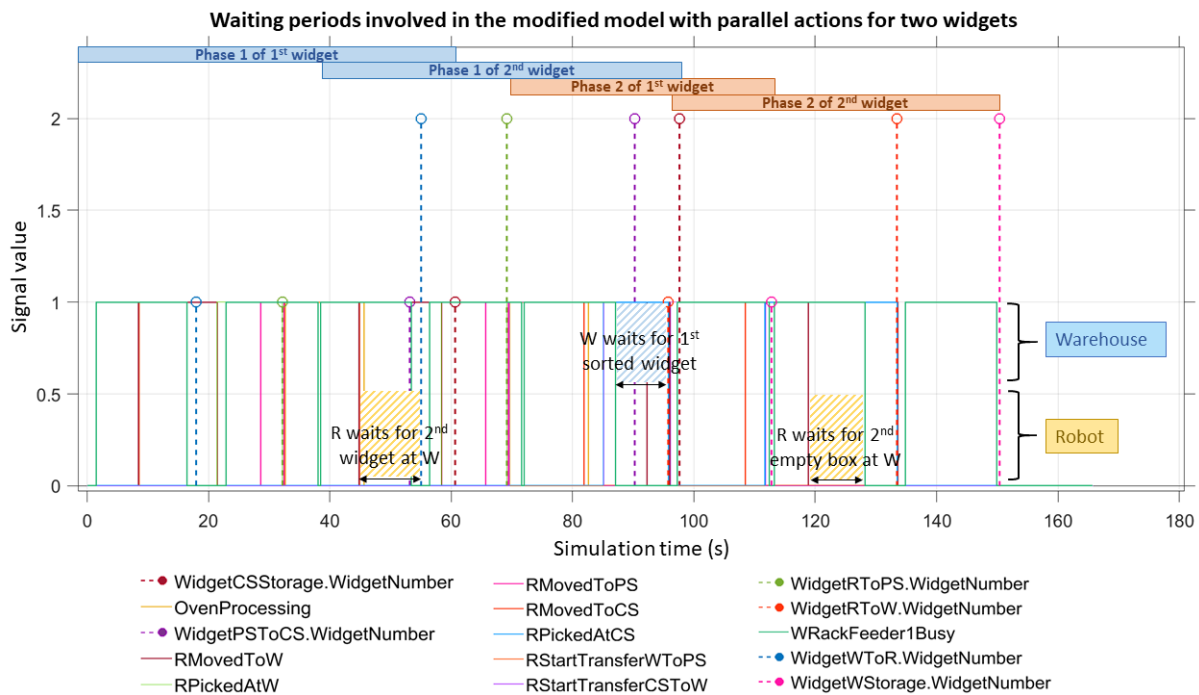


Figure 4.15: Time Graph highlighting the waiting periods involved in the modified model with parallel actions for two widgets in Phase-1 and Phase-2.

A few waiting periods could still be spotted in the working of Warehouse and Robot, and are highlighted in the graph shown in Figure 4.15. They were of two types:

- Warehouse waits to receive the sorted widget while the Robot is busy in getting the widget from Color Sorter.
- Robot waits at the Warehouse to pickup or place while Warehouse is busy in retrieving a full or empty box from the storage rack.

Through careful examination, it was realized that it was not possible to remove the remaining waiting periods with the default configuration of the factory model setup. There was a

need to introduce multiple components or subcomponents of the Warehouse and Robot in the factory model. These extra components would work in parallel to the existing components, thereby, ensuring that at least one of them is always available for performing the next action while the other one completes the previous action. In this project, the Warehouse component was modified in Simulink such that it consisted of two rack feeders working in parallel with each other. In this way, they could retrieve and store boxes consecutively to and fro the storage rack and conveyor belt at the Warehouse. This model is explained in the next section.

4.3 Modified model with two rack feeders

The modified Simulink subsystems and Stateflow charts of Supervisory Control and Warehouse are depicted in Appendix D. In the Warehouse subsystem, two Stateflow charts were added for each rack feeder and were connected to the main Stateflow chart of Warehouse. All the other subsystems were the same as in the previous model for parallel execution of independent actions.

Since the working of the Supervisory Control in this model was same as the previous model, it is not explained again in this section. The decision making process involved in the Warehouse subsystem with two rack feeders is demonstrated by the flow chart depicted in Figure 4.16. Here, W refers to the Warehouse, SC refers to the Supervisory Control and RF refers to rack feeder.

In Phase-1, W checks if there is a request from SC to retrieve a widget. If yes, then, W checks if RF1 is available. If yes, then RF1 is sent to retrieve a full box from the storage rack. Then, W checks if RF2 is free. If yes, then RF2 is sent to retrieve another full box from the rack. Even if RF1 is not available in the first place, it is checked if RF2 is free and the same steps are followed. W checks if RF1 and RF2 have retrieved full boxes successfully. If neither of them retrieves, then the W goes back to the step where it checks for the SC request and repeats the procedure. If RF1 or RF2 retrieve full boxes, W checks if the conveyor belt is free. If it is free, then priority is given to RF1 to place the full box. If RF1 has not retrieved a full box, then RF2 is allowed to place the full box on the belt. W informs SC that it is ready with a full box on the conveyor belt so that Robot can pickup the widget. W then keeps checking if the Robot has picked up the widget. If yes, then it will check if RF1 is free to store the empty box back to the storage rack. If RF1 is busy, it will check if RF2 is free to store the box. Either RF1 or RF2 stores the empty box at the storage rack. When one RF is busy in storing the empty box, W checks if the other RF has retrieved another full box to place at the conveyor belt. If yes, then it follows the same process as explained above. If not retrieved, then W goes back to the initial step and checks if there is a new request from SC to retrieve a widget. If yes, then, W directs the free RF to retrieve another full box from rack to the conveyor belt.

In Phase-2, a similar decision making process is followed for the storing of sorted widgets. RF1 and RF2 retrieve empty boxes from the storage rack to the conveyor belt. W waits for Robot to place widget into the empty box. RF1 and RF2 then store the full boxes back to the storage rack.

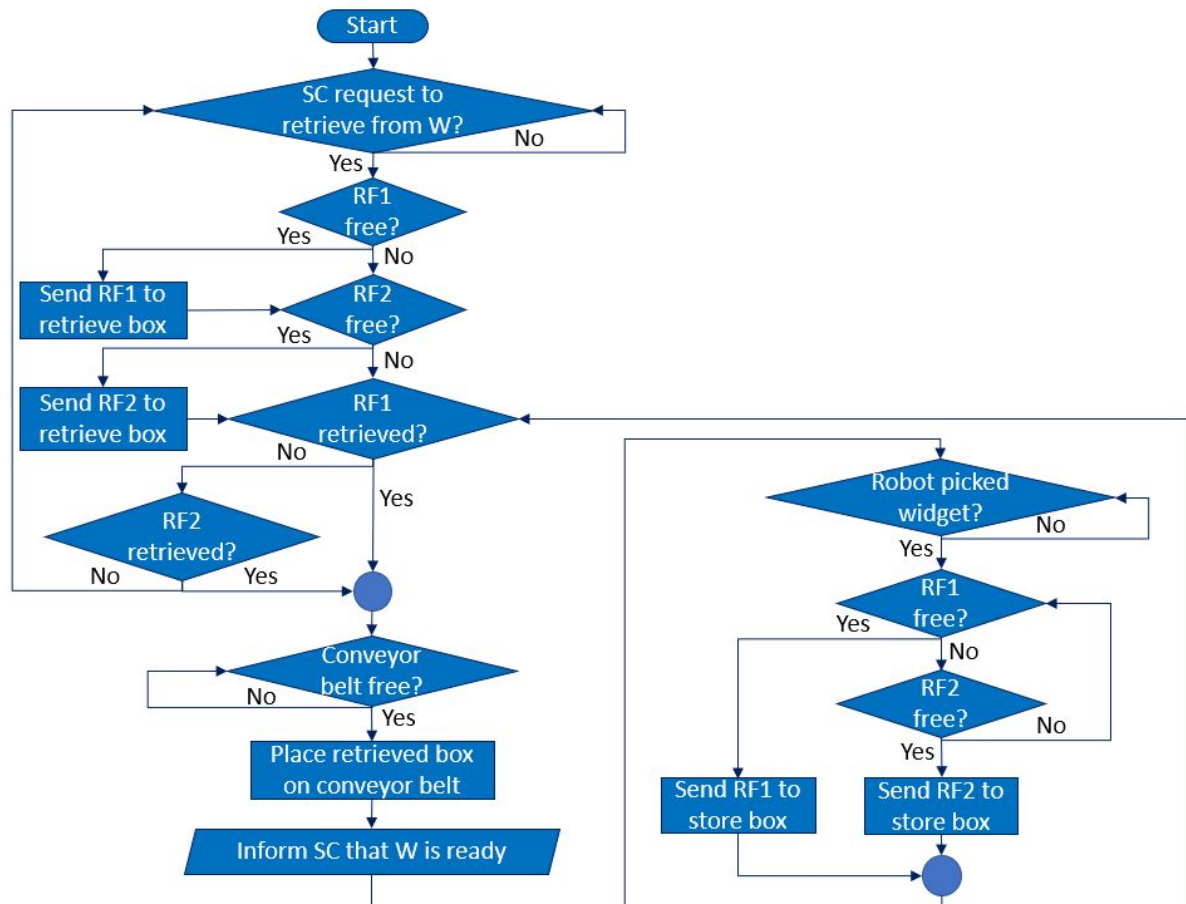


Figure 4.16: Flow chart showing the decision making process involved in the modified Warehouse subsystem with two rack feeders.

The overall working of the supervisory control in this model was similar to the previously modified model with parallel execution of independent actions. Hence, the signal value versus simulation time graph for single widget is not depicted for this model. The simulation process was carried out for all nine widgets in the modified model with two rack feeders at the Warehouse. This movement of widgets is visualized with the help of the graph shown in Figure 4.17. It can be observed that the second widget in Phase-1 (represented by second blue dotted line with circle top) is retrieved by the Warehouse very quickly after the first widget was retrieved (represented by first blue dotted line with circle top). This indicates that RF2 was ready with the second widget and placed it on the conveyor belt as soon as RF1 picked up the empty box of first widget. This process is followed

throughout the Phase-1 and Phase-2 where RF1 and RF2 work in parallel for retrieving and storing full or empty boxes.

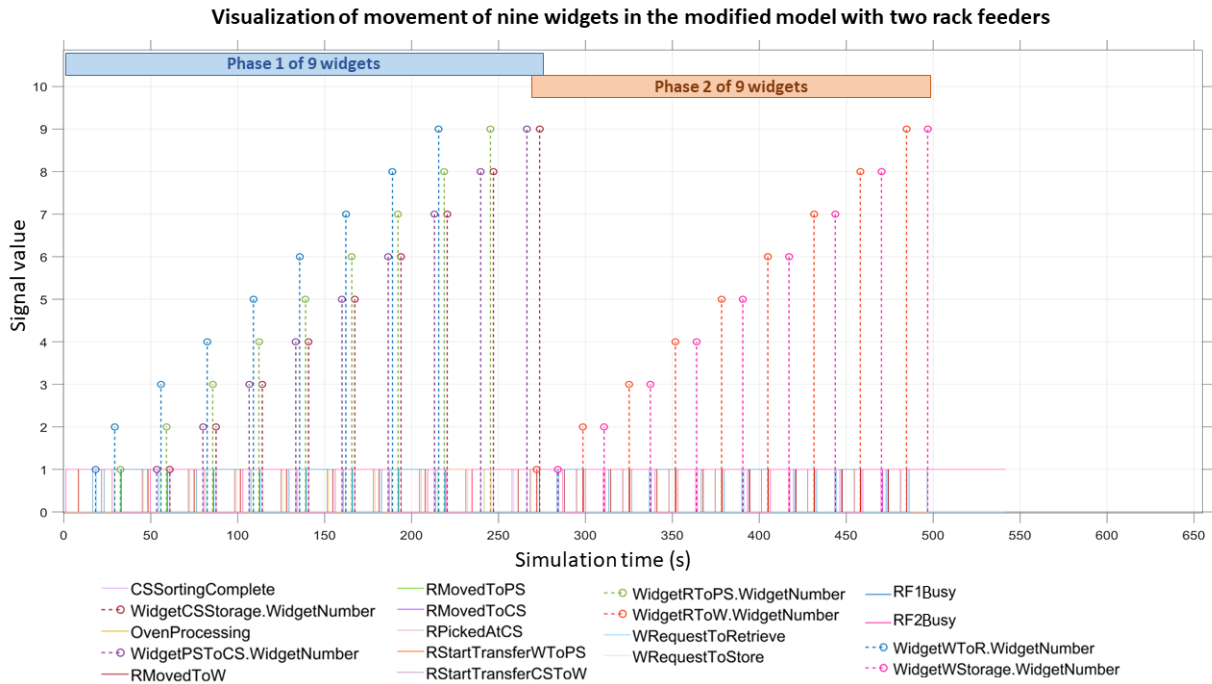


Figure 4.17: Graph for visualization of nine widgets moving through factory model components in Phase-1 and Phase-2 of the modified model with two rack feeders.

To closely examine this behavior, a simulation of this model was carried out with only two widgets present in the system. The important actions in Warehouse (W) and Robot (R) involved in this procedure are depicted in 4.18. The actions of RF1 and RF2 are also marked for the reader to understand. In addition, the remaining waiting periods in the working of the Warehouse are also highlighted. These waiting periods exist as the Robot is busy completing its previous action and hence, cannot reach the Warehouse to complete the next action. Therefore, an extra Robot component would be needed to remove these waiting periods from the working of the factory model.

In this implementation, MathWorks tools helped in modifying the system architecture of the factory model by adding another rack feeder in the Warehouse. Moreover, system analysis using MathWorks tools suggested that a Robot should also be added in order to optimize the system performance further. Hence, this implementation and its results answered the third research question of this project (RQ3).

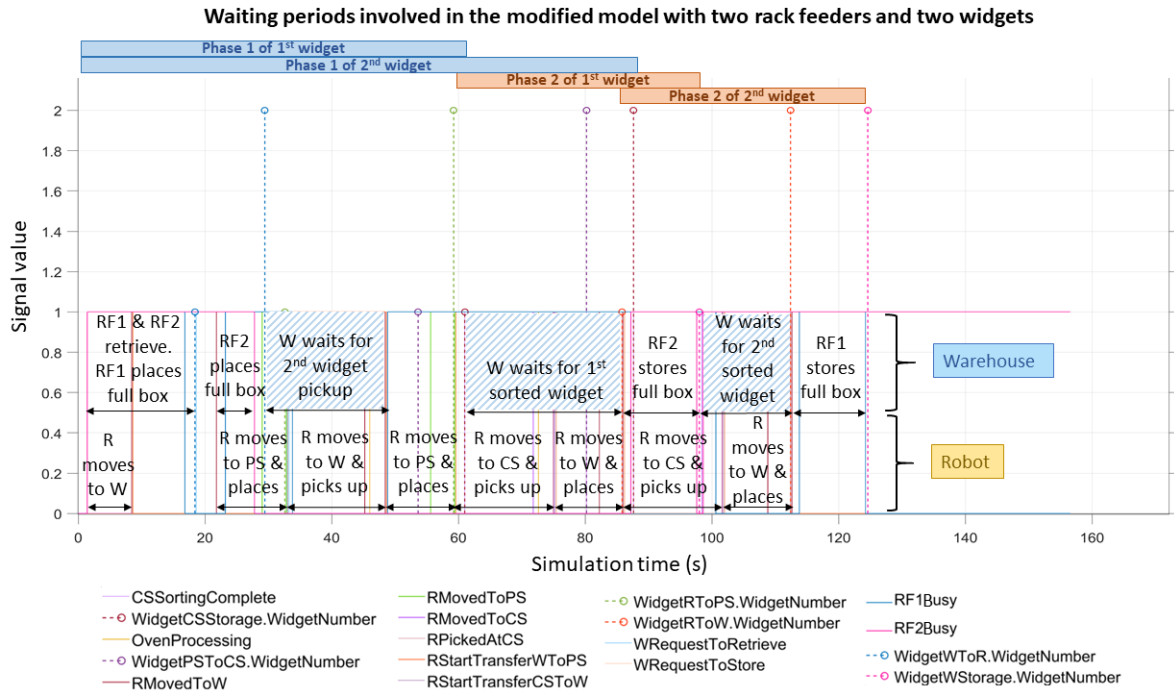


Figure 4.18: Time Graph highlighting the waiting periods involved in the modified model with two rack feeders and two widgets moving in Phase-1 and Phase-2.

Throughout the simulation processes of all the factory model implementations in Simulink, the Simulink function block calculated the system throughput and logged the values. One such example is shown in Figure 4.19. Using the throughput values logged in case of the benchmark model, a graph was plotted for the system throughput versus simulation time of ten hours. The simulation process included multiple iterations of movement of nine widgets in the factory model.

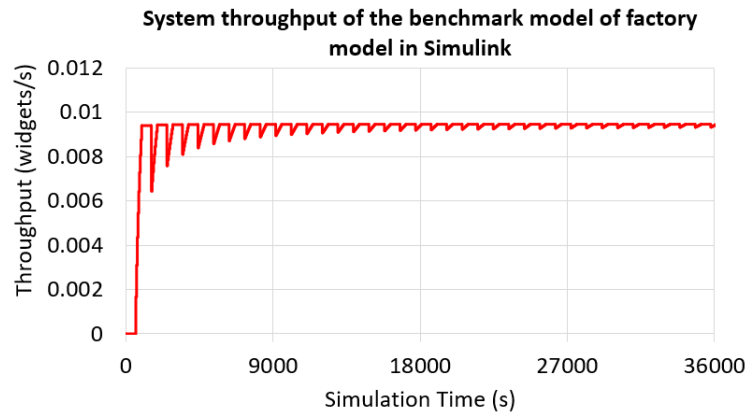


Figure 4.19: Graph of system throughput of the benchmark model of factory model measured in Simulink simulation run over a period of ten hours.

In the first iteration, the throughput was zero during Phase-1 as no widgets were stored in the Warehouse rack. As Phase-2 started, widgets were brought back to the storage rack one by one. Hence, the throughput increased and reached a maximum value when the ninth widget was stored. In Phase-1 of the second iteration, the Simulink function block did not calculate throughput as there were no new widgets being stored in the rack. Thus, the throughput value remained constant in the graph. As Phase-2 of the second iteration started, the Simulink function block also started calculating throughput again. By this point, the simulation time had increased significantly but the number of stored widgets had been the same as they were in last iteration. Hence, the throughput dropped low to a certain value. As more widgets were stored in the rack in Phase-2 of the second iteration, throughput again increased to the maximum value reached in the first iteration.

In the beginning of each subsequent iteration, the throughput was constant during Phase-1 and equal to the maximum value reached in the last iteration. Then, it dropped to a lower value as Phase-2 started and then, increased to the maximum possible value as more widgets were again stored in the Warehouse rack. This behavior can be clearly seen in Figure 4.19. As the number of stored widgets increased over a period of ten hours, the drops observed in throughput in the beginning of Phase-2 of each iteration became less significant.

The total execution time for one widget as well as nine widgets were measured for every implementation of the factory model in Simulink. Similarly, the system throughput was measured for all the implementations in Simulink and average throughput was calculated for each implementation. These values are shown in Table 4.1.

It was observed that the system performance of the factory model was improved with the modifications introduced to the benchmark model. The total execution time for moving one widget through the factory model was slightly decreased in the second implementation as compared to the first implementation. The total execution time for moving nine widgets was drastically reduced in the second implementation because of parallel execution of independent actions. As a result, the system throughput improved in the second implementation as compared to the first implementation.

As expected, the total execution time for one widget was nearly the same in the third implementation as that in the second implementation. Owing to similarity in the overall working of supervisory control in the two implementations, there could be very little change in the total execution time for one widget. Moreover, the computation and instructions were increased in case of the third implementation because of the complex decision-making involved in the working of the two rack feeders. However, there was a significant reduction in total execution time for nine widgets in the third implementation. Hence, an improvement could also be seen in the system throughput.

Table 4.1: Comparison of total execution time and average system throughput of the implementations of factory model in Simulink

| S.No. | Factory model implementation in Simulink | Total execution time for one widget (s) | Total execution time for nine widgets (s) | Average system throughput over a period of ten hours (widgets/s) |
|-------|--|---|---|--|
| 1 | Benchmark model (combination of sequential and parallel execution of actions) | 112 | 960 | 0.0083 |
| 2 | Modified model with parallel execution of independent actions | 92 | 649 | 0.0127 |
| 3 | Modified model with parallel execution of independent actions and two Warehouse rack feeders | 94 | 485 | 0.0164 |

As compared to the first implementation, the total execution time for nine widgets in the third implementation was reduced approximately by 49% (halved) and the system throughput was improved by 98% (doubled). Therefore, it was concluded that the third implementation was the best model of factory model in Simulink with optimized system performance. In this way, MathWorks tools helped in optimizing the overall performance of a system developed using MBSE. This answered the first research question of this project (RQ1).

4.4 Live-link between Simulink and Digital Twin

For publishing MQTT messages from Simulink to the Digital Twin in Prespective, the Warehouse subsystem in the benchmark model was modified as shown in the Figure 4.20. The highlighted red box shows that a MATLAB function block named as `MQTT_Wrapper` was added and the input to the block was a signal called `StartDigitalTwinRF` from the Warehouse Stateflow chart. The modified Stateflow chart of the Warehouse subsystem in benchmark model is included in Appendix E.

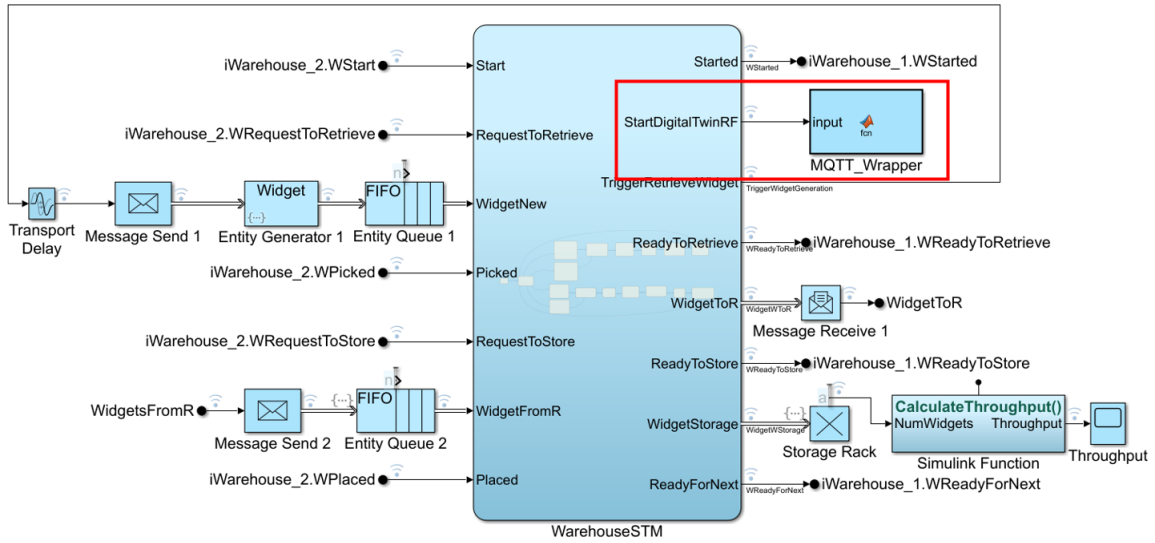


Figure 4.20: Inclusion of a MATLAB function block in the Warehouse subsystem of benchmark model

Using the MQTT toolbox in MATLAB, a MATLAB code was written and added in Simulink as a function named `MQTT_Wrapper`. The code is given as follows:

```

1 function MQTT_Wrapper(input)
2
3 %%Retain values in memory between function calls
4 persistent myMQTT;
5 persistent old_input;
6
7 %% Connect to MQTT broker only once
8 if isempty(myMQTT)
9     myMQTT = mqtt('tcp://localhost','ClientID','myClient','Port',1883)
10 end
11
12 %% Publish message to topic at broker if input value changes
13 new_input = input;
14 if (new_input ≠ old_input)
15     input_json = jsonencode(input);
16     publish(myMQTT, 'Warehouse_Simulink', input_json);
17 end
18 old_input = input;

```

Whenever the value of `StartDigitalTwinRF` signal was changed by the state actions in Warehouse Stateflow chart, a new message was published by the `MQTT_Wrapper` MATLAB function to the topic `Warehouse_Simulink` at the Eclipse Mosquitto broker. The broker address was specified by line 9 in the code. Using the Prespective plugin for MQTT, the

Digital Twin model was configured by the engineers from Prespective such that it also subscribed to the topic `Warehouse_Simulink` at the broker. Hence, the value received in the MQTT message was used to rotate the DC motor of the rack feeder in the Warehouse of Digital Twin. A negative integer value moved the rack feeder from the conveyor belt to the storage rack at the Warehouse whereas a positive integer value moved the rack feeder vice-versa. Figure 4.21 shows the simulation of benchmark model running in Simulink along with the Digital Twin running in Prespective.

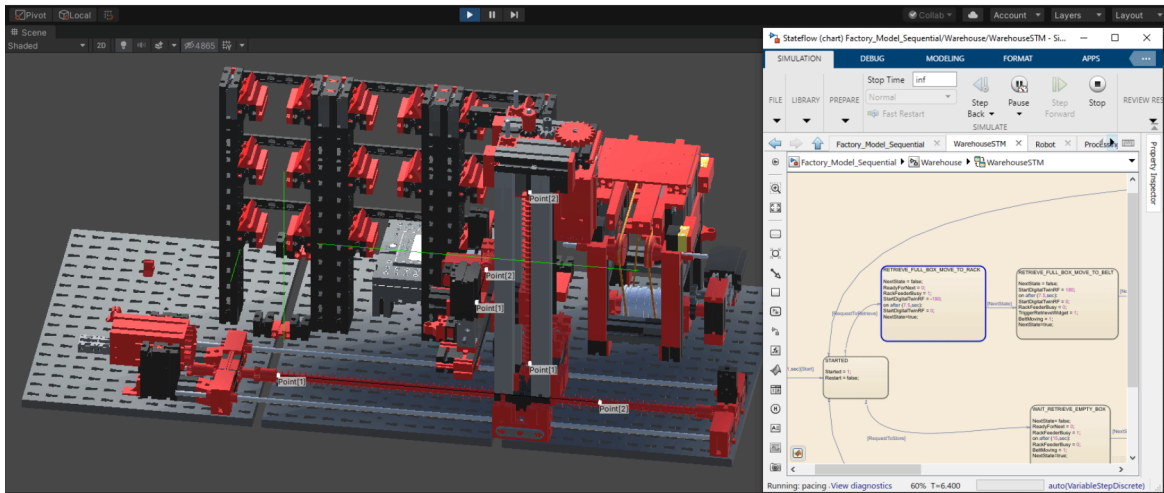


Figure 4.21: The rack feeder in the Digital Twin (subscriber) moved by sending MQTT messages from Simulink model (publisher).

In this way, a live-link between a Simulink model and a Digital Twin was successfully created using MQTT messaging protocol. Therefore, this implementation answered the fourth research question targeted in this project (RQ4).

Chapter 5

Conclusions and future work

The combination of MathWorks tools - Simulink, Stateflow, SimEvents and MATLAB, considered in this project proved to be beneficial in improving the existing design of a CPS developed using MBSE methodology. The supervisory-control of the Fischertechnik Factory Simulation 24V was successfully modelled, analyzed and modified using the MathWorks tools.

With the help of the modifications introduced to the factory model in Simulink, the total execution time for nine widgets was reduced to half of its value as in the original model. Moreover, the system throughput in the fully modified implementation was doubled as compared to the original model. Since the system performance of the factory model was optimized by using the chosen MathWork tools, the first research question (RQ1) of this project was successfully answered. The MathWork tools facilitated system analysis of the factory model in such a way that long waiting time periods were discovered. To utilize these waiting periods, the factory model was modified in Simulink such that independent actions involved in the system development process were executed in parallel at the same time. In this way, the second research question (RQ2) was answered in this project. Furthermore, MathWorks tools helped in simulating the addition of an extra rack feeder at the Warehouse of factory model. The system analysis of the modified model also suggested that an extra Robot component would be needed to further improve the system performance. Therefore, the role of MathWorks tools in the modification of factory model with an extra rack feeder and the suggestion to add another Robot represented a fitting answer to the third research question (RQ3) of this project. Lastly, MQTT networking protocol was used for communication between the Warehouse subsystem of benchmark model in Simulink and Digital Twin of factory model in Prespective. The establishment of this live-link answered the fourth research question (RQ4) of this project.

For future work, it is recommended that another Robot component be added to the factory model in Simulink. The first Robot could be used to execute Phase-1 of all widgets while

the second Robot could be used to execute Phase-2 of the widgets as soon as the first sorted widget is available at the Color Sorter. If the system analysis of this modified model reveals a new bottleneck at the Warehouse, then, it is recommended to add another conveyor belt at the Warehouse of the factory model in Simulink. The factory model could be configured such that the first set of Warehouse rack feeder, conveyor belt and Robot are responsible for carrying out Phase-1 of all the widgets while the second set of these components are responsible for carrying out Phase-2 of the sorted widgets. The Digital Twin of the factory model could be further improved to enable the movement of virtual widgets. Then, the modifications to the factory model suggested by the Simulink models in this project could be validated by implementing the same in Digital Twin. Moreover, the Simulink model of the factory model could be configured to facilitate MQTT subscribe such that messages from the Digital Twin could be received in Simulink via the broker. After testing and validation with the Digital Twin is complete, the aspects of the modified system design that are found to be feasible could be applied on the physical implementation of the Fischertechnik Factory Simulation 24V.

References

- [1] M. Chen, “Industry 4.0: A deep drive into the future of factory,” 2020. [Online]. Available: <https://oosga.com/en/industry4-0/>
- [2] N. Jankevicius, “SysML model integration with MATLAB/Simulink,” 2015. [Online]. Available: <https://www.nomagic.com/events/webinars/item/webinar-6-2015-sysml-model-integration-with-matlab-simulink%C2%AE>
- [3] VitechCorporation, “Simulink Connector Guide,” 2018. [Online]. Available: <http://www.vitechcorp.com/support/documentation/genesys/600/SimulinkConnectorGuide.pdf>
- [4] Fischertechnik, “Factory Simulation 24V.” [Online]. Available: <https://www.fischertechnik.de/en/service/elearning/simulating/fabrik-simulation-24v>
- [5] Deloitte, “Industry 4.0 and the Digital Twin,” 2020. [Online]. Available: <https://www2.deloitte.com/cn/en/pages/consumer-industrial-products/articles/industry-4-0-and-the-digital-twin.html>
- [6] i SCOOP, “Industry 4.0: the fourth industrial revolution – guide to Industrie 4.0,” 2020. [Online]. Available: <https://www.i-scoop.eu/industry-4-0/>
- [7] Y. Lu, “Cyber Physical System (CPS)-based Industry 4.0: A survey,” *Journal of Industrial Integration and Management*, vol. 02, p. 1750014, 11 2017.
- [8] EuropeanCommission, “Cyber - Physical Systems,” 2019. [Online]. Available: <https://ec.europa.eu/digital-single-market/en/cyber-physical-systems>
- [9] M. Törngren and U. Sellgren, “Complexity challenges in development of cyber-physical systems,” in *Lecture Notes in Computer Science*. Springer International Publishing, 2018, pp. 478–503. [Online]. Available: https://doi.org/10.1007/978-3-319-95246-8_27
- [10] J. P. Wade, R. S. Cohen, N. S. Bowen, and E. Hole, “Systems Engineering of Cyber-Physical Systems: An Integrated Education Program,” in *ASEE’s 123rd Annual Conference & Exposition, New Orleans, LA, USA, June 26–29, 2016*. Washington D.C., USA: ASEE, 2016.

-
- [11] INCOSE, *Systems Engineering Handbook*, 4th ed. New Jersey, USA: John Wiley & Sons, 2015.
- [12] A. M. Madni and M. Sievers, “Model-based systems engineering: Motivation, current status, and research opportunities,” *Systems Engineering*, vol. 21, no. 3, pp. 172–190, 2018. [Online]. Available: <https://onlinelibrary.wiley.com/doi/abs/10.1002/sys.21438>
- [13] IBM, “Overview of Rational Rhapsody,” 2020. [Online]. Available: https://www.ibm.com/support/knowledgecenter/SSB2MU_8.3.0/com.ibm.rhp.overview.doc/topics/rhp_c_po_rr_product_overview.html
- [14] NoMagic, “Cameo Systems Modeler ,” 2020. [Online]. Available: <https://www.nomagic.com/products/cameo-systems-modeler#intro>
- [15] VitechCorporation, “Integrated software tools,” 2020. [Online]. Available: https://www.vitechcorp.com/?page_id=31160
- [16] MathWorks, “Math. Graphics. Programming.” 2020. [Online]. Available: <https://www.mathworks.com/products/matlab.html>
- [17] —, “Simulation and Model-Based Design,” 2020. [Online]. Available: <https://www.mathworks.com/products/simulink.html>
- [18] IBM, “Integrating Rational Rhapsody and the MathWorks Simulink,” 2020. [Online]. Available: https://www.ibm.com/support/knowledgecenter/SSB2MU_8.3.0/com.ibm.rhp.integ.designtools.doc/topics/rhp_c_int_rhp_and_simulink.html
- [19] L. Xing-hua and C. Yun-feng, “Design of UAV flight control system virtual prototype using Rhapsody and Simulink,” in *2010 International Conference On Computer Design and Applications*, 2010, pp. 34–38.
- [20] T. Sakairi, E. Palachi, C. Cohen, Y. Hatsutori, J. Shimizu, and H. Miyashita, “Model based control system design using SysML, Simulink, and Computer Algebra System,” *Journal of Control Science and Engineering*, vol. 2013.
- [21] S. Dasgupta, “Integration with MATLAB,” 2019. [Online]. Available: <https://docs.nomagic.com/display/CST185/Integration+with+MATLAB>
- [22] A. Nitsch, B. Beichler, F. Golatowski, and C. Haubelt, “Model-based systems engineering with MATLAB/Simulink in the railway sector,” in *Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen (MBMV)*, 2015.
- [23] MathWorks, “Model-based design of cyber-physical systems in MATLAB and Simulink,” 2020. [Online]. Available: <https://www.mathworks.com/discovery/cyber-physical-systems.html>

REFERENCES

- [24] —, “Model and simulate decision logic using state machines and flow charts,” 2020. [Online]. Available: <https://www.mathworks.com/products/stateflow.html>
- [25] —, “Model and simulate message communication and discrete-event systems,” 2020. [Online]. Available: <https://www.mathworks.com/products/simevents.html>
- [26] —, “Design and analyze system and software architectures,” 2020. [Online]. Available: <https://nl.mathworks.com/products/system-composer.html>
- [27] —, “MATLAB and Simulink for Model-Based Systems Engineering,” 2020. [Online]. Available: <https://www.mathworks.com/solutions/model-based-systems-engineering.html>
- [28] Verum, “Discover Dezyne - Formal verification & model driven development,” 2019. [Online]. Available: <https://www.verum.com/dezyne/>
- [29] BeagleBoard.org, “BeagleBone Black.” [Online]. Available: <https://beagleboard.org/black>
- [30] Maplesoft, “Industry 4.0 and the power of the Digital Twin,” 2020. [Online]. Available: <https://www.maplesoft.com/ns/manufacturing/industry-4-0-power-of-the-digital-twin.aspx>
- [31] Prespective, “Merging virtual and physical worlds,” 2020. [Online]. Available: <https://prespective-software.com/perspective/>
- [32] OASIS, “MQTT: The standard for IoT messaging,” 2020. [Online]. Available: <https://mqtt.org/>
- [33] R. Sanvordenker, “Visualization and testing of an autonomously driving truck’s SysML models in a virtual 3D simulation environment,” Master’s thesis, Eindhoven University of Technology, 2020.
- [34] MathWorks, “Compose and analyze a system,” 2020. [Online]. Available: <https://www.mathworks.com/help/systemcomposer/gs/compose-and-analyze-a-system.html>
- [35] —, “Publish MQTT Messages and Subscribe to Message Topics,” 2020. [Online]. Available: <https://www.mathworks.com/help/supportpkg/raspberrypi/ref/publish-and-subscribe-to-mqtt-messages.html>
- [36] —, “MQTT in MATLAB,” 2020. [Online]. Available: <https://www.mathworks.com/matlabcentral/fileexchange/64303-mqtt-in-matlab>
- [37] —, “Create custom functionality using MATLAB function block,” 2020. [Online]. Available: <https://www.mathworks.com/help/simulink/ug/creating-an-example-model-that-uses-a-matlab-function-block.html>

- [38] —, “Control Chart Execution by Using Temporal Logic,” 2020. [Online]. Available: <https://www.mathworks.com/help/stateflow/ug/using-temporal-logic-in-state-actions-and-transitions.html>
- [39] —, “View Data in the Simulation Data Inspector,” 2020. [Online]. Available: <https://www.mathworks.com/help/simulink/ug/populate-sdi-with-your-data.html>
- [40] —, “Function defined with Simulink blocks,” 2020. [Online]. Available: <https://www.mathworks.com/help/simulink/slref/simulinkfunction.html>
- [41] Eclipse, “Eclipse Mosquitto - An open source MQTT broker,” 2020. [Online]. Available: <https://mosquitto.org/>

Appendix A

Architecture model in System Composer

The system architecture of the factory model as modelled in System Composer is depicted in Figures A.1 and Figures A.2. It consisted of five components named as SupervisoryControl, Warehouse, Robot, ProcessingStation and ColorSorter. The interfaces were named as iWarehouse, iRobot, iProcessingStation and iColorSorter.

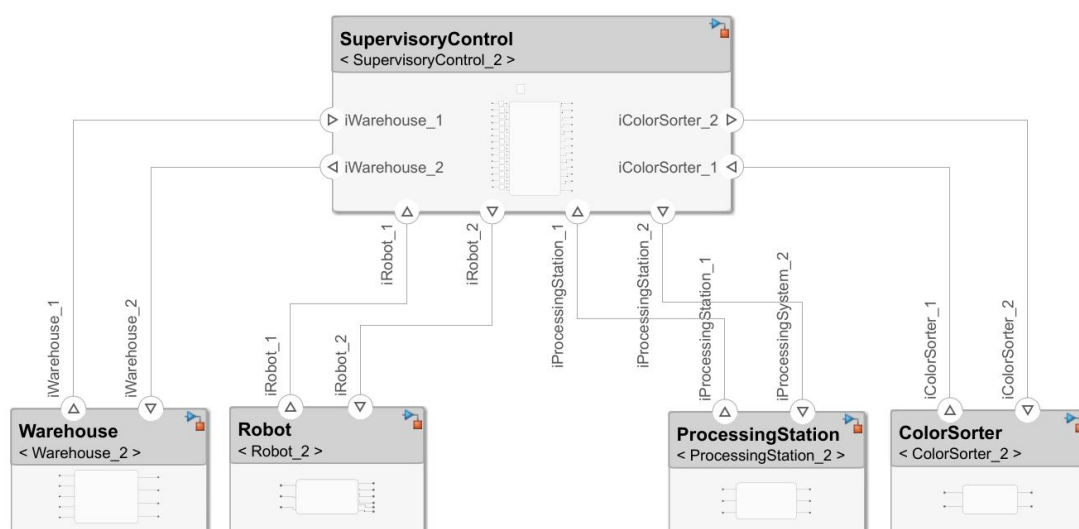


Figure A.1: Components in the architecture model of the factory model in System Composer.

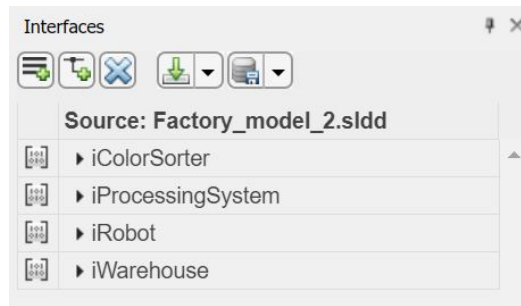


Figure A.2: Interfaces in the architecture model of the factory model in System Composer.

Appendix B

Benchmark model

B.1 Warehouse subsystem

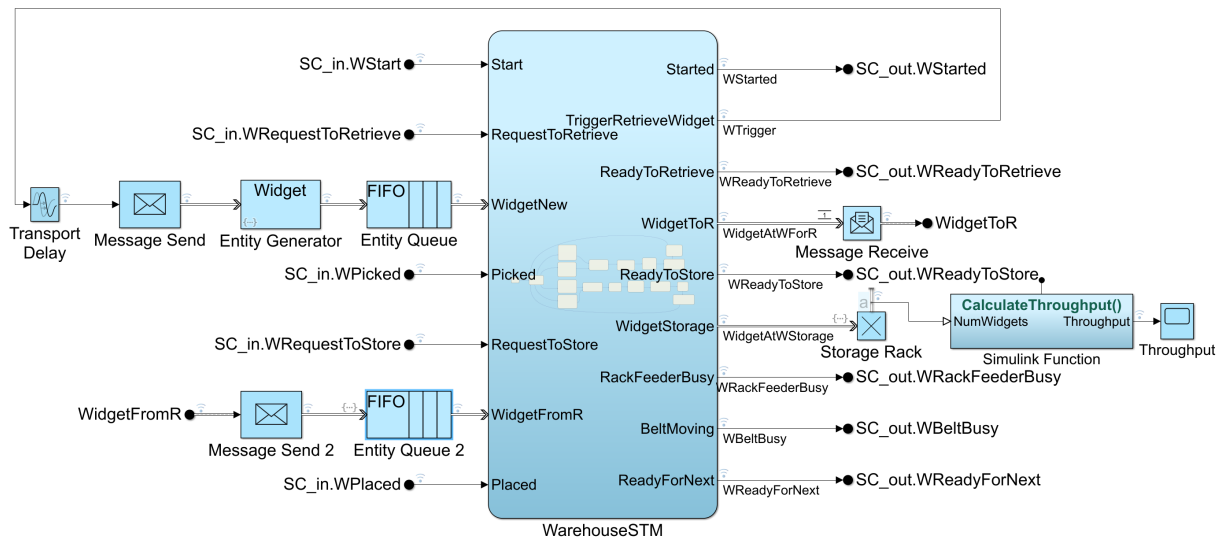


Figure B.1: Inside the Warehouse subsystem in Simulink.



















| | Name | Scope | Port | DataType | InitialValue |
|---|-----------------------|--------|------|-------------|--------------|
|  | Start | Input | 1 | double | |
|  | RequestToRetrieve | Input | 2 | double | |
|  | WidgetNew | Input | 3 | Bus: Widget | |
|  | Picked | Input | 4 | double | |
|  | RequestToStore | Input | 5 | double | |
|  | WidgetFromR | Input | 6 | Bus: Widget | |
|  | Placed | Input | 7 | double | |
|  | NextState | Local | | boolean | false |
|  | AtRack | Local | | boolean | false |
|  | Started | Output | 1 | double | 0 |
|  | TriggerRetrieveWidget | Output | 2 | double | 0 |
|  | ReadyToRetrieve | Output | 3 | double | 0 |
|  | WidgetToR | Output | 4 | Bus: Widget | |
|  | ReadyToStore | Output | 5 | double | 0 |
|  | WidgetStorage | Output | 6 | Bus: Widget | |
|  | RackFeederBusy | Output | 7 | double | 0 |
|  | BeltMoving | Output | 8 | double | 0 |
|  | ReadyForNext | Output | 9 | double | 0 |

Figure B.2: Information of input and output signals used in the Warehouse Stateflow chart (from Simulink Model Explorer).

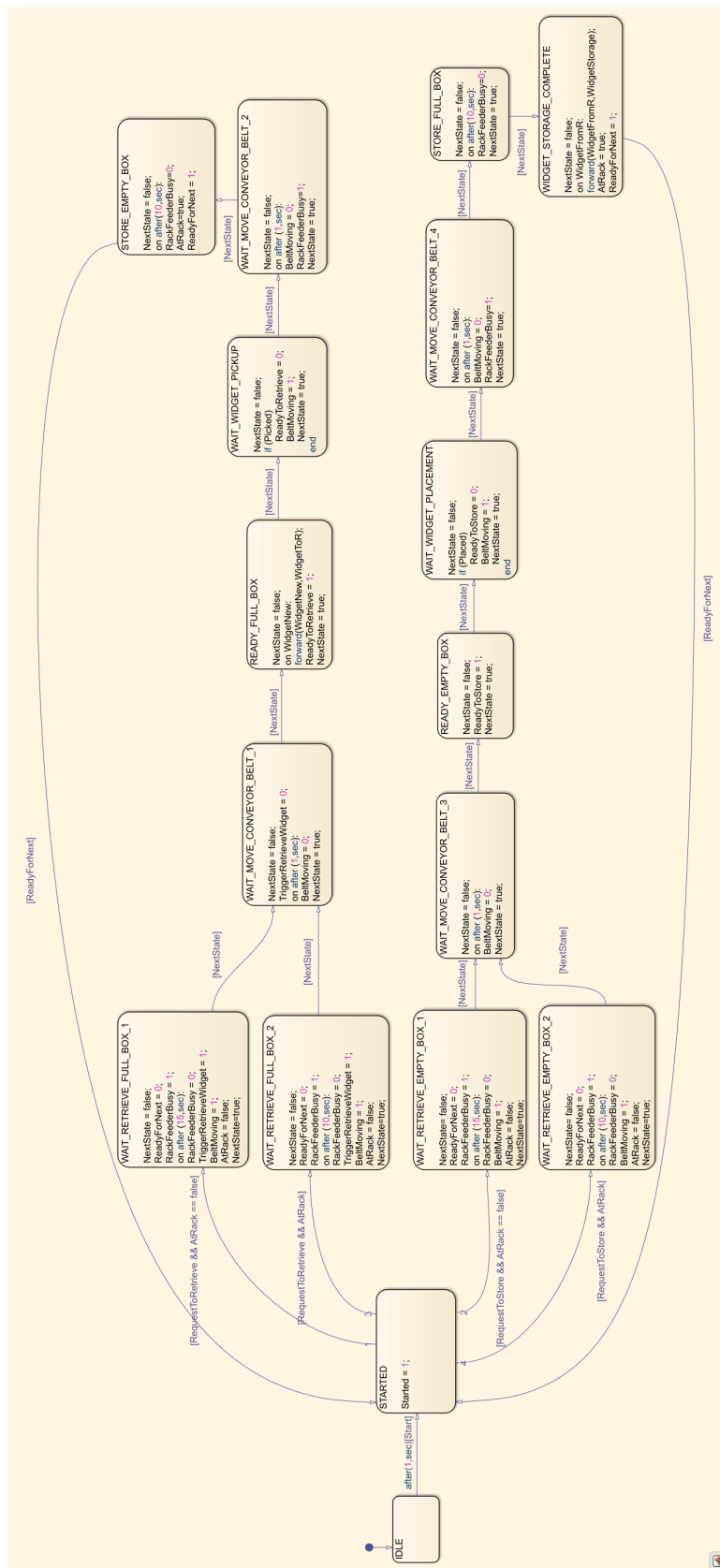


Figure B.3: Inside the Warehouse Stateflow chart.

B.2 Robot subsystem

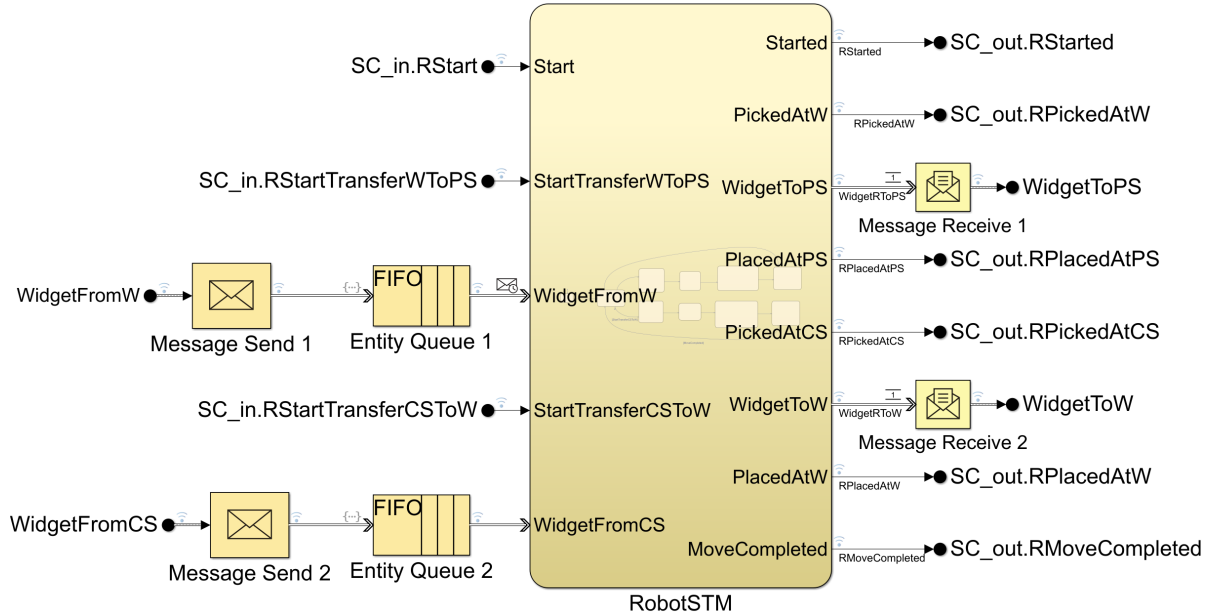


Figure B.4: Inside the Robot subsystem in Simulink.

| Name | Scope | Port | DataType | InitialValue |
|--------------------|--------|------|-------------|--------------|
| Start | Input | 1 | double | |
| StartTransferWToPS | Input | 2 | double | |
| WidgetFromW | Input | 3 | Bus: Widget | |
| StartTransferCSToW | Input | 4 | double | |
| WidgetFromCS | Input | 5 | Bus: Widget | |
| NextState | Local | | boolean | false |
| Started | Output | 1 | double | 0 |
| PickedAtW | Output | 2 | double | 0 |
| WidgetToPS | Output | 3 | Bus: Widget | |
| PlacedAtPS | Output | 4 | double | 0 |
| PickedAtCS | Output | 5 | double | 0 |
| WidgetToW | Output | 6 | Bus: Widget | |
| PlacedAtW | Output | 7 | double | 0 |
| MoveCompleted | Output | 8 | double | 0 |

Figure B.5: Information of input and output signals used in the Robot Stateflow chart (from Simulink Model Explorer).

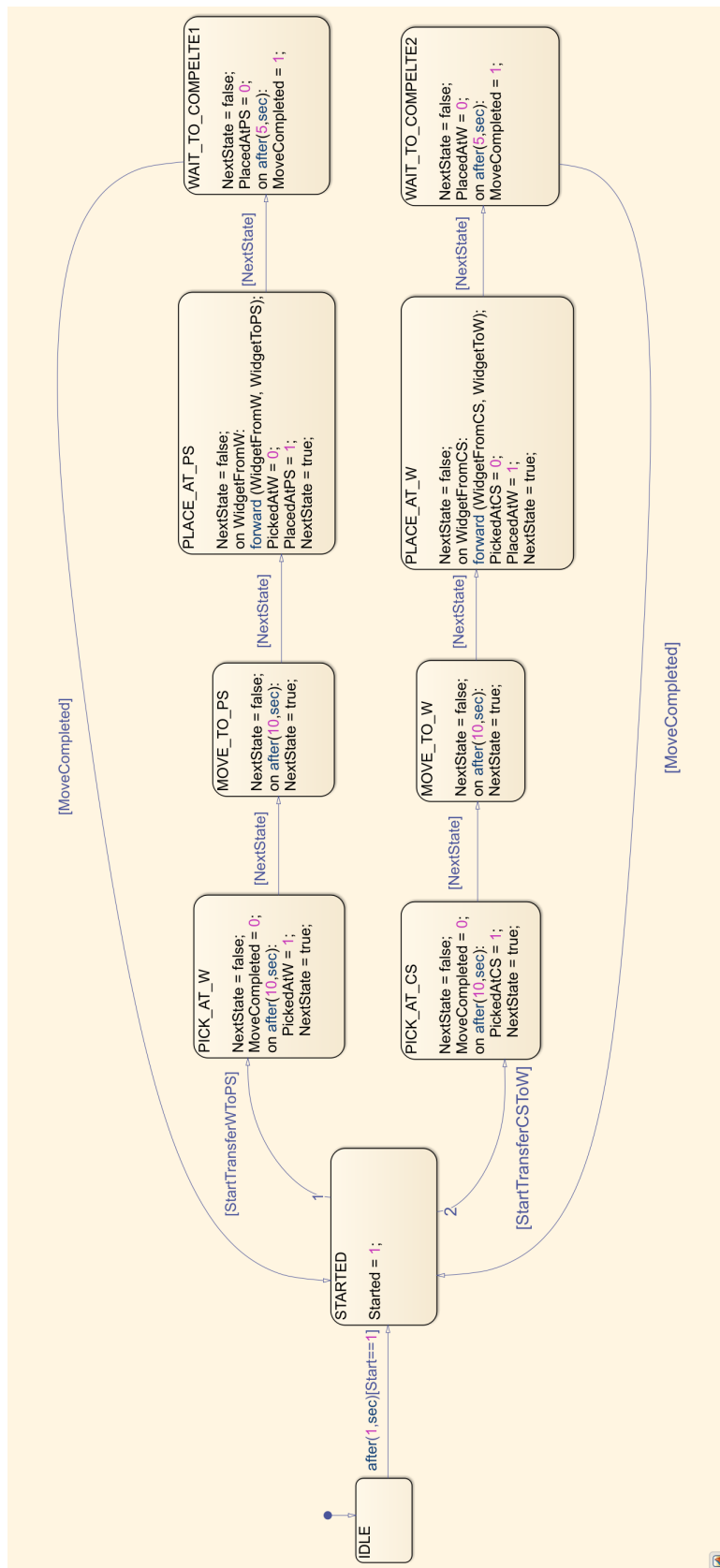


Figure B.6: Inside the Robot Stateflow chart.

B.3 Processing Station subsystem

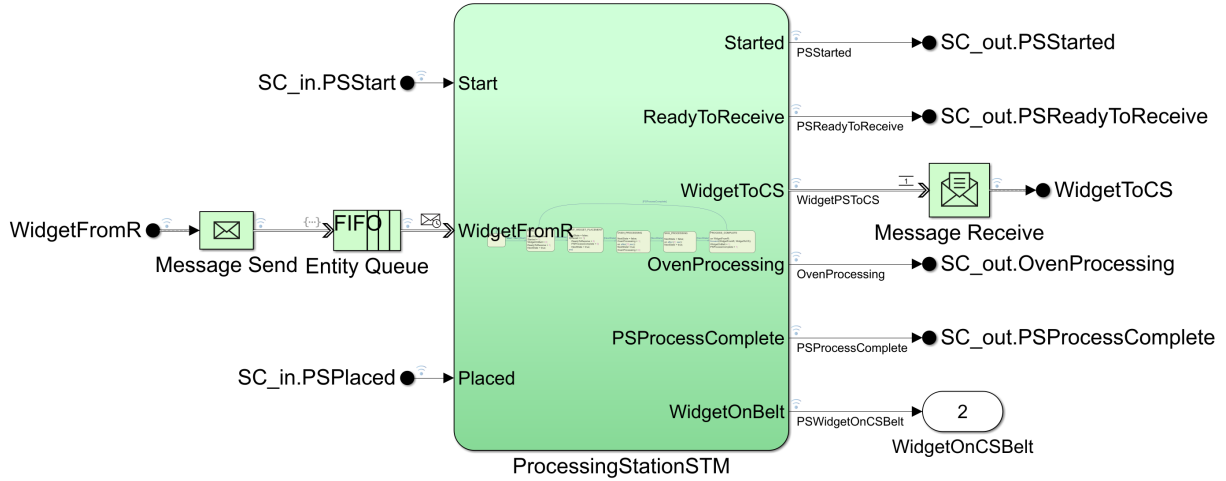


Figure B.7: Inside the Processing Station subsystem in Simulink.

| Name | Scope | Port | Data Type | Initial Value |
|-------------------|--------|------|-------------|---------------|
| Start | Input | 1 | double | |
| WidgetFromR | Input | 2 | Bus: Widget | |
| Placed | Input | 3 | double | |
| NextState | Local | | boolean | false |
| Started | Output | 1 | double | 0 |
| ReadyToReceive | Output | 2 | double | 0 |
| WidgetToCS | Output | 3 | Bus: Widget | |
| OvenProcessing | Output | 4 | double | 0 |
| PSProcessComplete | Output | 5 | double | 0 |
| WidgetOnBelt | Output | 6 | double | 0 |

Figure B.8: Information of input and output signals used in the Processing Station State-flow chart (from Simulink Model Explorer).

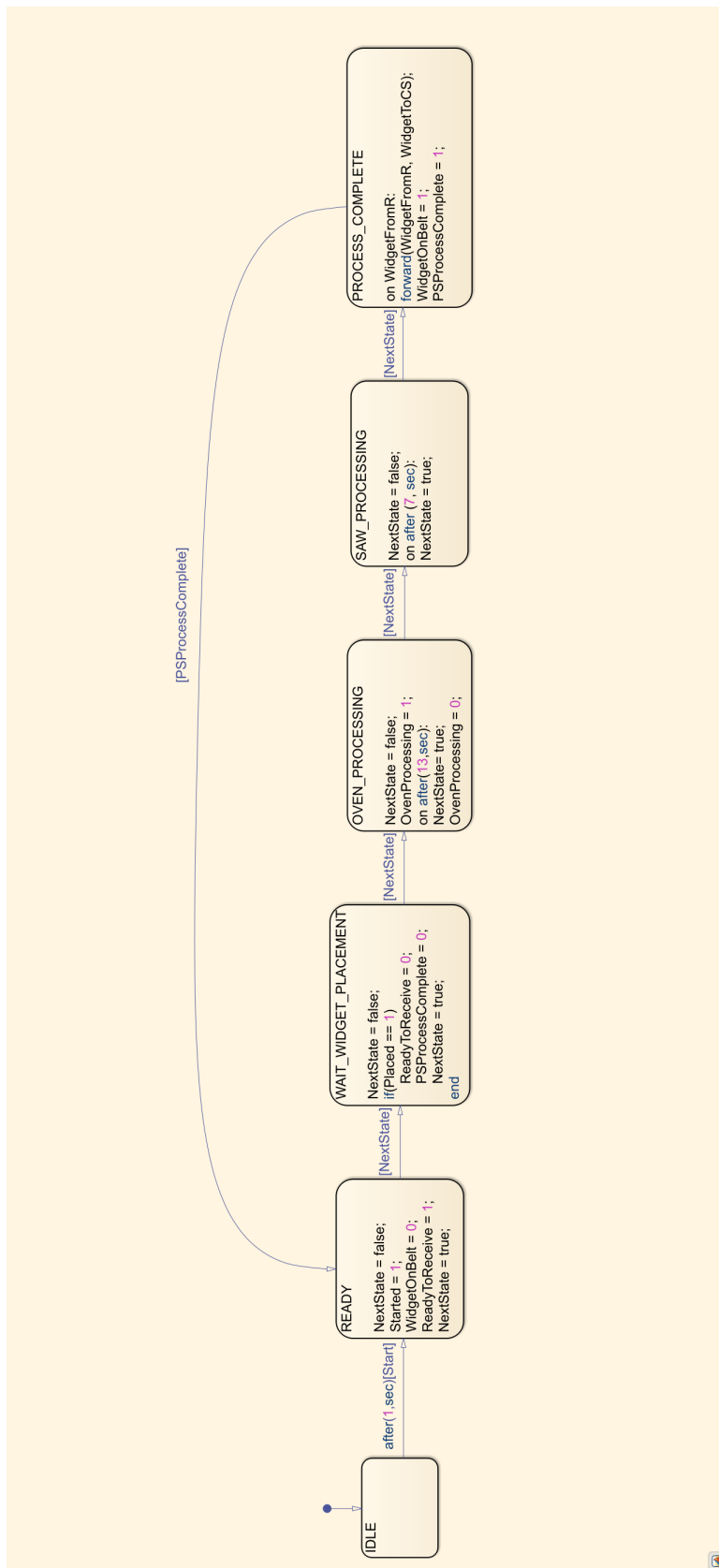


Figure B.9: Inside the Processing Station Stateflow chart.

B.4 Color Sorter subsystem

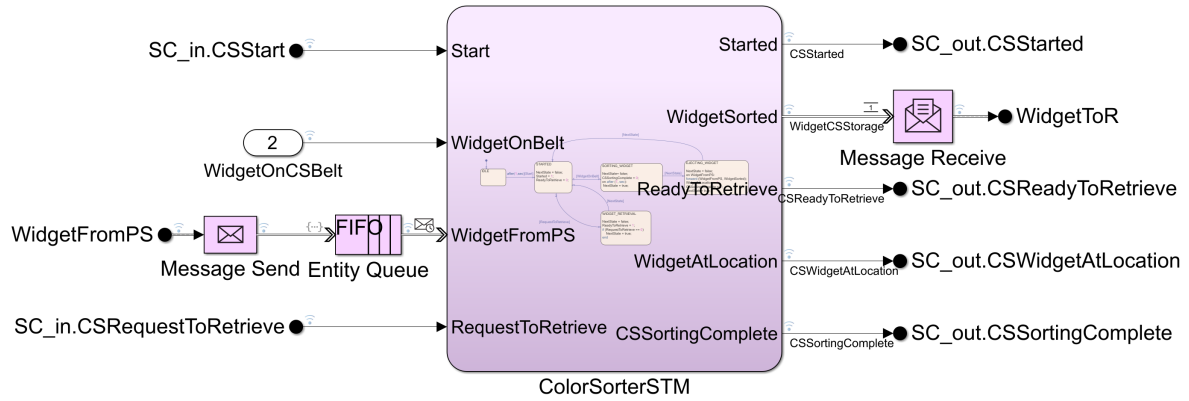


Figure B.10: Inside the Color Sorter subsystem in Simulink.

| | Name | Scope | Port | Data Type | Initial Value |
|--|-------------------|--------|------|-------------|---------------|
| | Start | Input | 1 | double | |
| | WidgetOnBelt | Input | 2 | double | |
| | WidgetFromPS | Input | 3 | Bus: Widget | |
| | RequestToRetrieve | Input | 4 | double | |
| | NextState | Local | | boolean | false |
| | Started | Output | 1 | double | 0 |
| | WidgetSorted | Output | 2 | Bus: Widget | |
| | ReadyToRetrieve | Output | 3 | double | 0 |
| | WidgetAtLocation | Output | 4 | double | 0 |
| | CSSortingComplete | Output | 5 | double | 0 |

Figure B.11: Information of input and output signals used in the Color Sorter Stateflow chart (from Simulink Model Explorer).

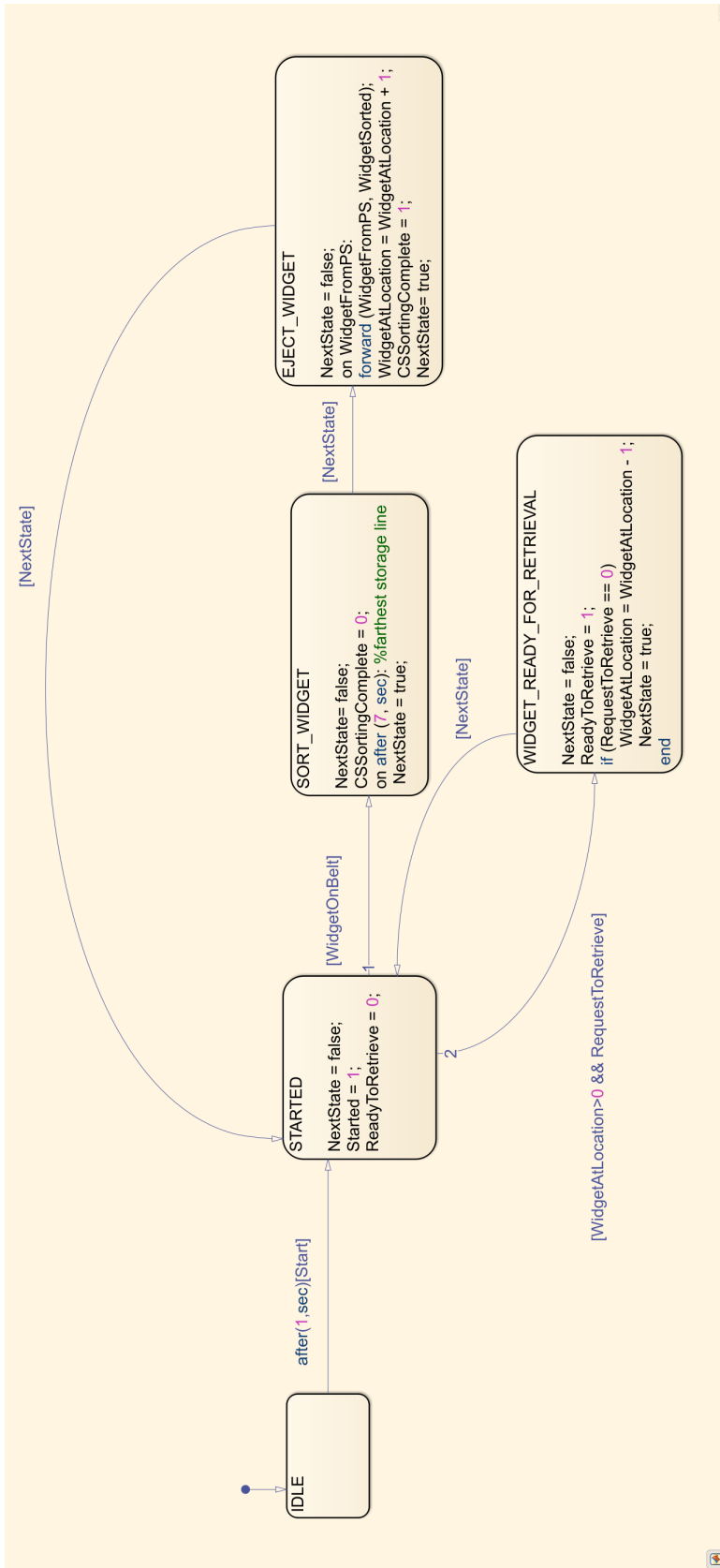


Figure B.12: Inside the Color Sorter Stateflow chart.

Appendix C

Modified model for parallel execution of independent actions

C.1 Modifications in the Supervisory Control subsystem



Figure C.1: Inside the Supervisory Control subsystem modified for parallel actions.

APPENDIX C. MODIFIED MODEL FOR PARALLEL EXECUTION OF INDEPENDENT ACTIONS

| | Name | Scope | Port | Data Type | Initial Value |
|---------|---------------------|--------|------|-----------|---------------|
| 101 010 | StartedWarehouse | Input | 1 | double | |
| 101 010 | StartedRobot | Input | 2 | double | |
| 101 010 | StartedPS | Input | 3 | double | |
| 101 010 | StartedCS | Input | 4 | double | |
| 101 010 | WReadyToRetrieve | Input | 5 | double | |
| 101 010 | PSReadyToReceive | Input | 6 | double | |
| 101 010 | RMovedToW | Input | 7 | double | |
| 101 010 | RPickedAtW | Input | 8 | double | |
| 101 010 | RMovedToPS | Input | 9 | double | |
| 101 010 | RPlacedAtPS | Input | 10 | double | |
| 101 010 | WReadyForNext | Input | 11 | double | |
| 101 010 | RMoveCompleted | Input | 12 | double | |
| 101 010 | OvenProcessing | Input | 13 | double | |
| 101 010 | PSProcessComplete | Input | 14 | double | |
| 101 010 | CSSortingComplete | Input | 15 | double | |
| 101 010 | WReadyToStore | Input | 16 | double | |
| 101 010 | CSReadyToRetrieve | Input | 17 | double | |
| 101 010 | RMovedToCS | Input | 18 | double | |
| 101 010 | RPickedAtCS | Input | 19 | double | |
| 101 010 | RPlacedAtW | Input | 20 | double | |
| 101 010 | NextState | Local | | boolean | false |
| 101 010 | Count1 | Local | | double | 0 |
| 101 010 | Count2 | Local | | double | 0 |
| 101 010 | TransferWToPS | Local | | boolean | false |
| 101 010 | TransferCSToW | Local | | boolean | false |
| 101 010 | x | Local | | double | 0 |
| 101 010 | StartWarehouse | Output | 1 | double | 0 |
| 101 010 | StartRobot | Output | 2 | double | 0 |
| 101 010 | StartPS | Output | 3 | double | 0 |
| 101 010 | StartCS | Output | 4 | double | 0 |
| 101 010 | WRequestToRetrieve | Output | 5 | double | 0 |
| 101 010 | RStartTransferWToPS | Output | 6 | double | 0 |
| 101 010 | RPickupAtW | Output | 7 | double | 0 |
| 101 010 | WPicked | Output | 8 | double | 0 |
| 101 010 | RPlaceAtPS | Output | 9 | double | 0 |
| 101 010 | PSPlaced | Output | 10 | double | 0 |
| 101 010 | WRequestToStore | Output | 11 | double | 0 |
| 101 010 | CSRequestToRetrieve | Output | 12 | double | 0 |
| 101 010 | RStartTransferCSToW | Output | 13 | double | 0 |
| 101 010 | RPickupAtCS | Output | 14 | double | 0 |
| 101 010 | CSPicked | Output | 15 | double | 0 |
| 101 010 | RPlaceAtW | Output | 16 | double | 0 |
| 101 010 | WPlaced | Output | 17 | double | 0 |

Figure C.2: Information of input and output signals used in the Supervisory Control Stateflow chart modified for parallel actions (from Simulink Model Explorer).

APPENDIX C. MODIFIED MODEL FOR PARALLEL EXECUTION OF INDEPENDENT ACTIONS

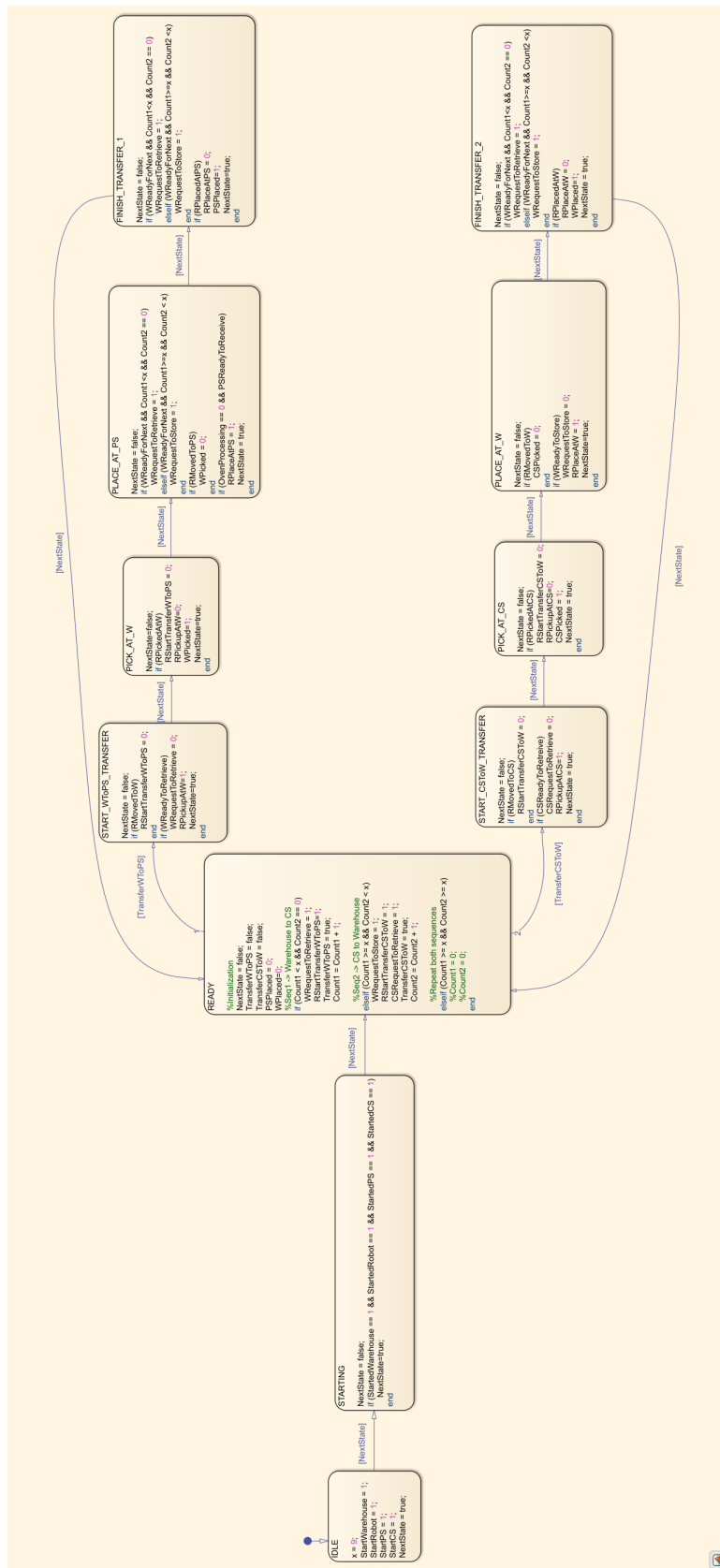


Figure C.3: Inside the Supervisory Control Stateflow chart modified for parallel actions.

C.2 Modifications in the Robot subsystem

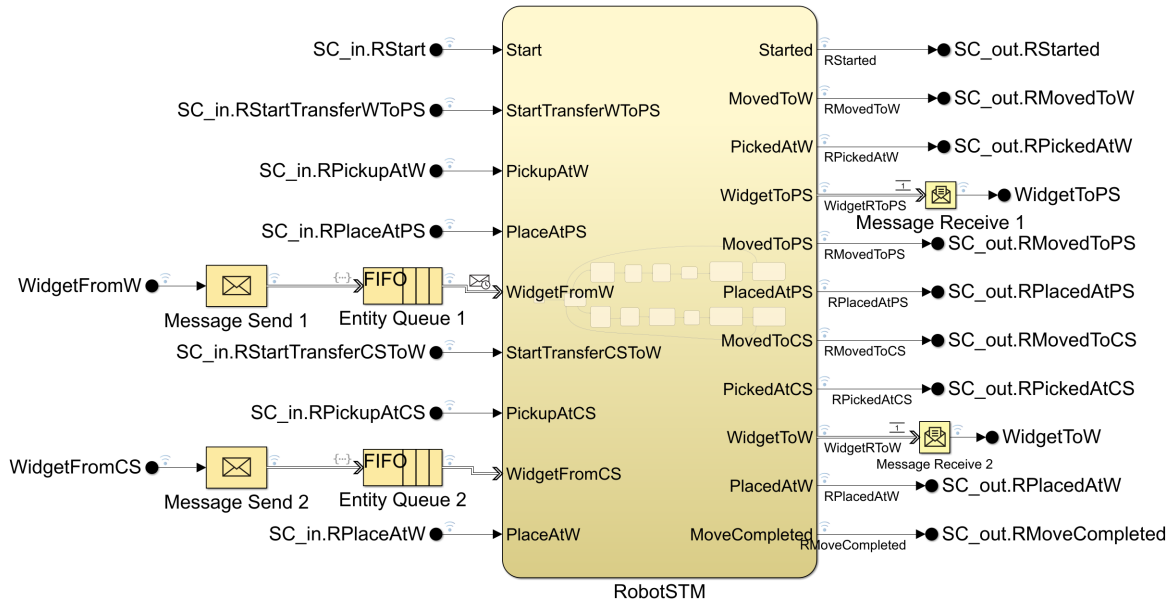


Figure C.4: Inside the Robot subsystem modified for parallel actions.

| Name | Scope | Port | DataType | InitialValue |
|--------------------|--------|------|-------------|--------------|
| Start | Input | 1 | double | |
| StartTransferWToPS | Input | 2 | double | |
| PickupAtW | Input | 3 | double | |
| PlaceAtPS | Input | 4 | double | |
| WidgetFromW | Input | 5 | Bus: Widget | |
| StartTransferCSToW | Input | 6 | double | |
| PickupAtCS | Input | 7 | double | |
| WidgetFromCS | Input | 8 | Bus: Widget | |
| PlaceAtW | Input | 9 | double | |
| NextState | Local | | boolean | false |
| Started | Output | 1 | double | 0 |
| MovedToW | Output | 2 | double | 0 |
| PickedAtW | Output | 3 | double | 0 |
| WidgetToPS | Output | 4 | Bus: Widget | |
| MovedToPS | Output | 5 | double | 0 |
| PlacedAtPS | Output | 6 | double | 0 |
| MovedToCS | Output | 7 | double | 0 |
| PickedAtCS | Output | 8 | double | 0 |
| WidgetToW | Output | 9 | Bus: Widget | |
| PlacedAtW | Output | 10 | double | 0 |
| MoveCompleted | Output | 11 | double | 0 |

Figure C.5: Information of input and output signals used in the Robot Stateflow chart modified for parallel actions (from Simulink Model Explorer).

APPENDIX C. MODIFIED MODEL FOR PARALLEL EXECUTION OF INDEPENDENT ACTIONS

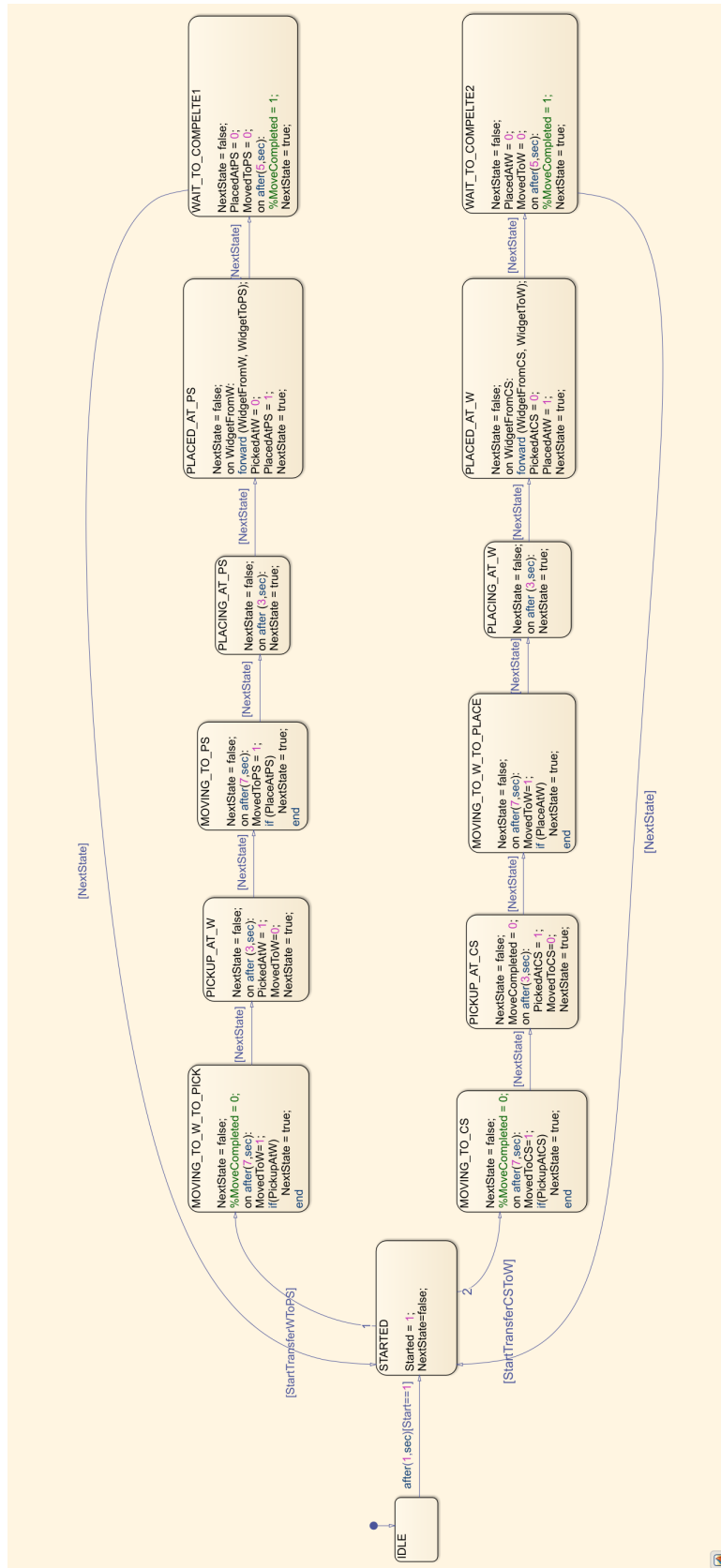


Figure C.6: Inside the Robot Stateflow chart modified for parallel actions.

Appendix D

Modified model with two rack feeders

D.1 Modifications in the Supervisory Control subsystem

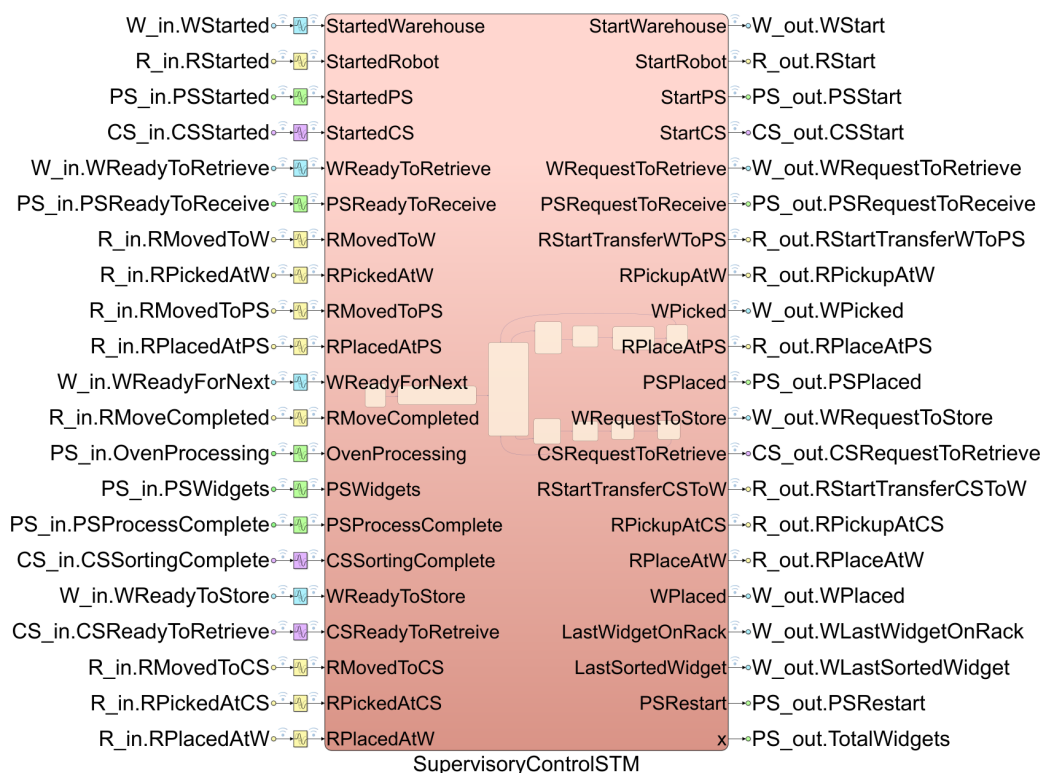


Figure D.1: Inside the Supervisory Control subsystem modified for two rack feeders at Warehouse.

APPENDIX D. MODIFIED MODEL WITH TWO RACK FEEDERS

















































| | Name | Scope | Port | Data Type | Initial Value |
|---|---------------------|--------|------|-----------|---------------|
|  | StartedWarehouse | Input | 1 | double | |
|  | StartedRobot | Input | 2 | double | |
|  | StartedPS | Input | 3 | double | |
|  | StartedCS | Input | 4 | double | |
|  | WReadyToRetrieve | Input | 5 | double | |
|  | PSReadyToReceive | Input | 6 | double | |
|  | RMovedToW | Input | 7 | double | |
|  | RPickedAtW | Input | 8 | double | |
|  | RMovedToPS | Input | 9 | double | |
|  | RPlacedAtPS | Input | 10 | double | |
|  | WReadyForNext | Input | 11 | double | |
|  | RMoveCompleted | Input | 12 | double | |
|  | OvenProcessing | Input | 13 | double | |
|  | PSWidgets | Input | 14 | double | |
|  | PSProcessComplete | Input | 15 | double | |
|  | CSSortingComplete | Input | 16 | double | |
|  | WReadyToStore | Input | 17 | double | |
|  | CSReadyToRetrieve | Input | 18 | double | |
|  | RMovedToCS | Input | 19 | double | |
|  | RPickedAtCS | Input | 20 | double | |
|  | RPlacedAtW | Input | 21 | double | |
|  | NextState | Local | | boolean | false |
|  | Count1 | Local | | double | 0 |
|  | Count2 | Local | | double | 0 |
|  | TransferWToPS | Local | | boolean | false |
|  | TransferCSToW | Local | | boolean | false |
|  | Done | Local | | double | 0 |
|  | StartWarehouse | Output | 1 | double | 0 |
|  | StartRobot | Output | 2 | double | 0 |
|  | StartPS | Output | 3 | double | 0 |
|  | StartCS | Output | 4 | double | 0 |
|  | WRequestToRetrieve | Output | 5 | double | 0 |
|  | PSRequestToReceive | Output | 6 | double | 0 |
|  | RStartTransferWToPS | Output | 7 | double | 0 |
|  | RPickupAtW | Output | 8 | double | 0 |
|  | WPicked | Output | 9 | double | 0 |
|  | RPlaceAtPS | Output | 10 | double | 0 |
|  | PSPlaced | Output | 11 | double | 0 |
|  | WRequestToStore | Output | 12 | double | 0 |
|  | CSRequestToRetrieve | Output | 13 | double | 0 |
|  | RStartTransferCSToW | Output | 14 | double | 0 |
|  | RPickupAtCS | Output | 15 | double | 0 |
|  | RPlaceAtW | Output | 16 | double | 0 |
|  | WPlaced | Output | 17 | double | 0 |
|  | LastWidgetOnRack | Output | 18 | double | 0 |
|  | LastSortedWidget | Output | 19 | double | 0 |
|  | PSRestart | Output | 20 | double | 0 |
|  | x | Output | 21 | double | 0 |

Figure D.2: Information of input and output signals used in the Supervisory Control State-flow chart modified for two rack feeders at Warehouse (from Simulink Model Explorer).

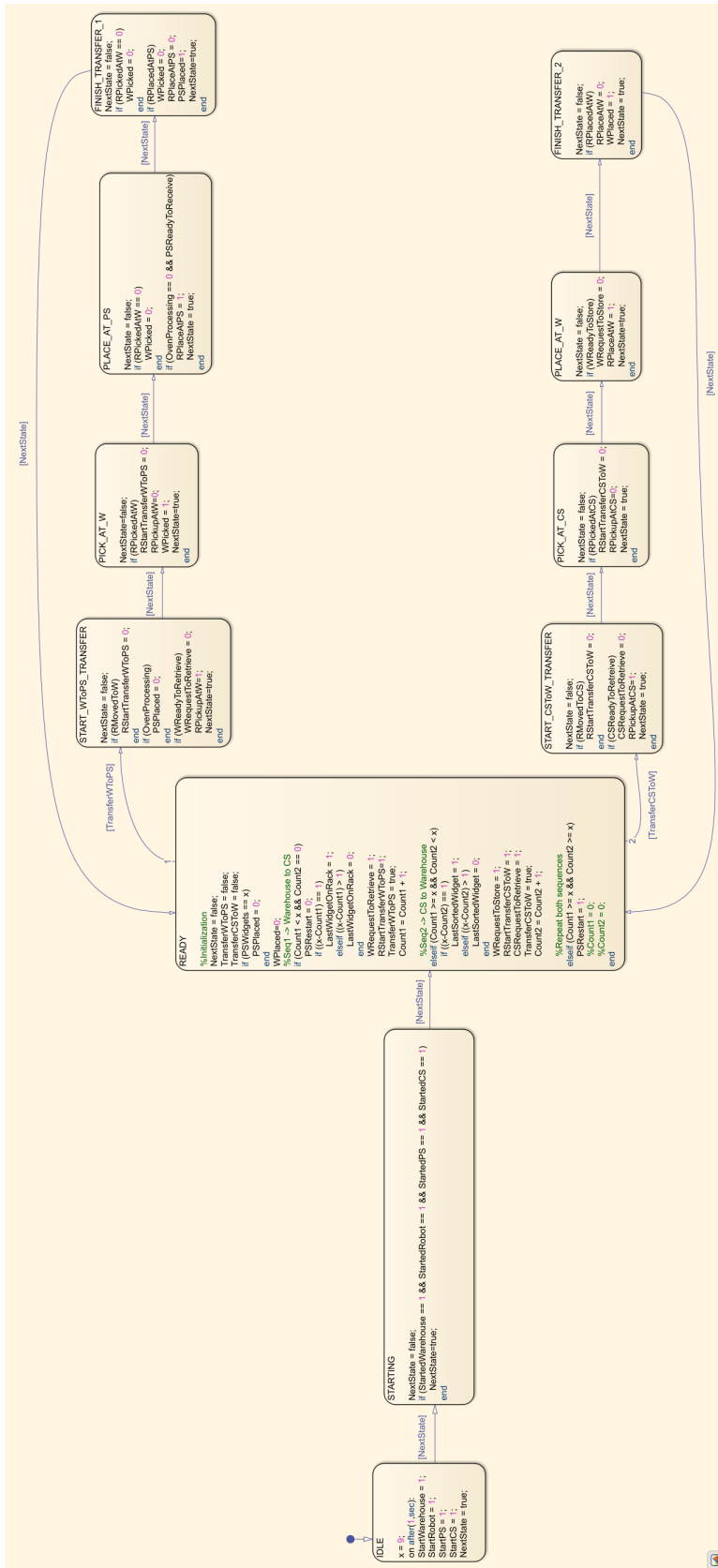


Figure D.3: Inside the Supervisory Control Stateflow chart modified for two rack feeders at Warehouse.

D.2 Modifications in the Warehouse subsystem

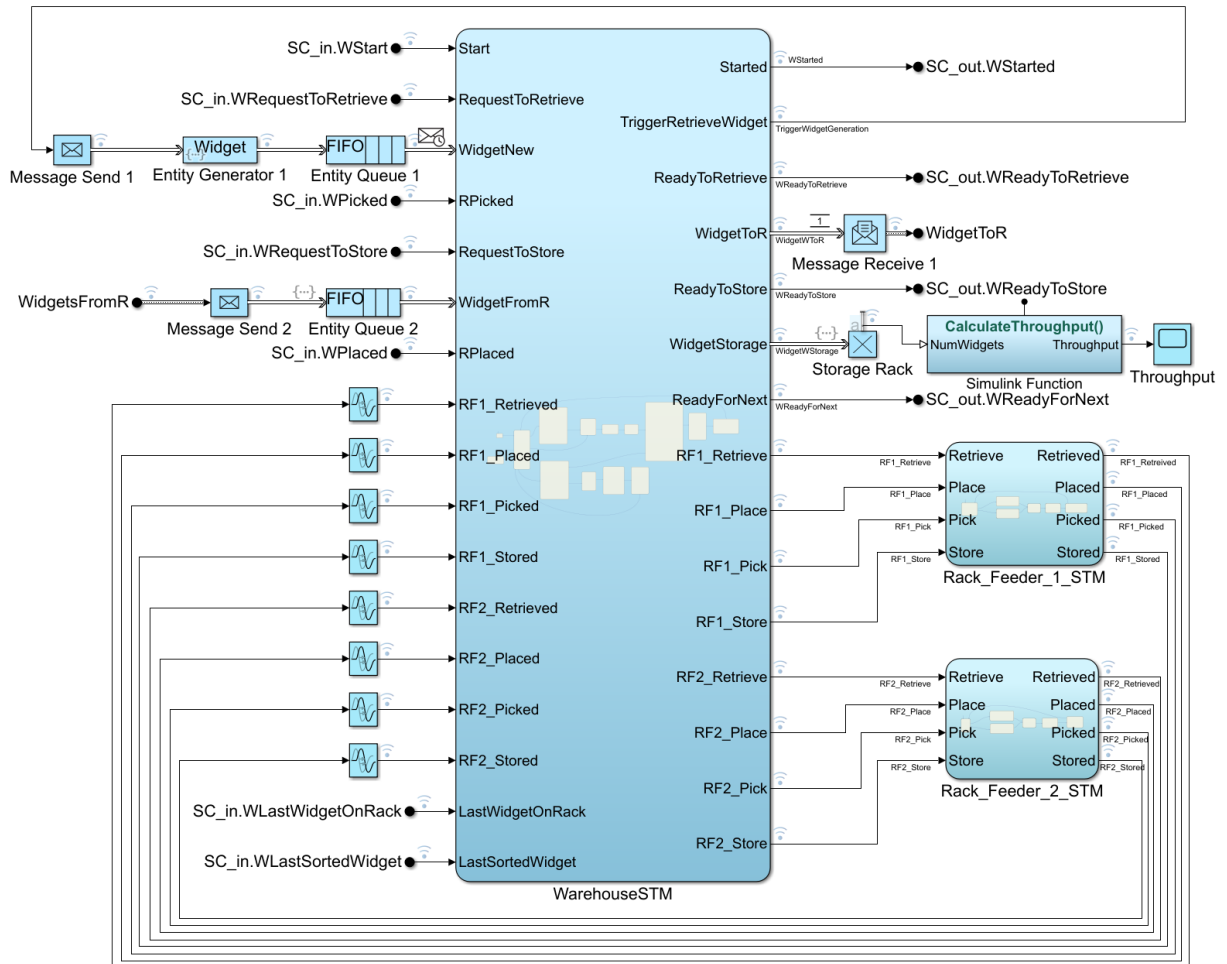


Figure D.4: Inside the Warehouse subsystem (modified for two rack feeders).

| | Name | Scope | Port | Data Type | Initial Value |
|--|-----------|--------|------|-----------|---------------|
| | Retrieve | Input | 1 | double | |
| | Place | Input | 2 | double | |
| | Pick | Input | 3 | double | |
| | Store | Input | 4 | double | |
| | NextState | Local | | boolean | false |
| | AtRack | Local | | boolean | false |
| | Retrieved | Output | 1 | double | 0 |
| | Placed | Output | 2 | double | 0 |
| | Picked | Output | 3 | double | 0 |
| | Stored | Output | 4 | double | 0 |

Figure D.5: Information of input and output signals used in the Rack Feeder Stateflow charts (from Simulink Model Explorer).

APPENDIX D. MODIFIED MODEL WITH TWO RACK FEEDERS

| | Name | Scope | Port | DataType | InitialValue |
|--|-----------------------|--------|------|-------------|--------------|
| | Start | Input | 1 | double | . |
| | RequestToRetrieve | Input | 2 | double | . |
| | WidgetNew | Input | 3 | Bus: Widget | . |
| | RPicked | Input | 4 | double | . |
| | RequestToStore | Input | 5 | double | . |
| | WidgetFromR | Input | 6 | Bus: Widget | . |
| | RPlaced | Input | 7 | double | . |
| | RF1_Retrieved | Input | 8 | double | . |
| | RF1_Placed | Input | 9 | double | . |
| | RF1_Picked | Input | 10 | double | . |
| | RF1_Stored | Input | 11 | double | . |
| | RF2_Retrieved | Input | 12 | double | . |
| | RF2_Placed | Input | 13 | double | . |
| | RF2_Picked | Input | 14 | double | . |
| | RF2_Stored | Input | 15 | double | . |
| | LastWidgetOnRack | Input | 16 | double | . |
| | LastSortedWidget | Input | 17 | double | . |
| | NextState | Local | | boolean | false |
| | RF2_Phase1 | Local | | double | 0 |
| | RF2_Phase2 | Local | | double | 0 |
| | Message | Local | | double | 0 |
| | RF1_Phase1 | Local | | double | 0 |
| | RF1_Phase2 | Local | | double | 0 |
| | PhaseTransition | Local | | double | 0 |
| | RF1_Busy | Local | | double | 0 |
| | BeltBusy | Local | | double | 0 |
| | RF2_Busy | Local | | double | 0 |
| | BeltMoving | Local | | double | 0 |
| | Started | Output | 1 | double | 0 |
| | TriggerRetrieveWidget | Output | 2 | double | 0 |
| | ReadyToRetrieve | Output | 3 | double | 0 |
| | WidgetToR | Output | 4 | Bus: Widget | . |
| | ReadyToStore | Output | 5 | double | 0 |
| | WidgetStorage | Output | 6 | Bus: Widget | . |
| | ReadyForNext | Output | 7 | double | 0 |
| | RF1_Retrieve | Output | 8 | double | 0 |
| | RF1_Place | Output | 9 | double | 0 |
| | RF1_Pick | Output | 10 | double | 0 |
| | RF1_Store | Output | 11 | double | 0 |
| | RF2_Retrieve | Output | 12 | double | 0 |
| | RF2_Place | Output | 13 | double | 0 |
| | RF2_Pick | Output | 14 | double | 0 |
| | RF2_Store | Output | 15 | double | 0 |

Figure D.6: Information of input and output signals used in the Warehouse Stateflow chart modified for two rack feeders (from Simulink Model Explorer).

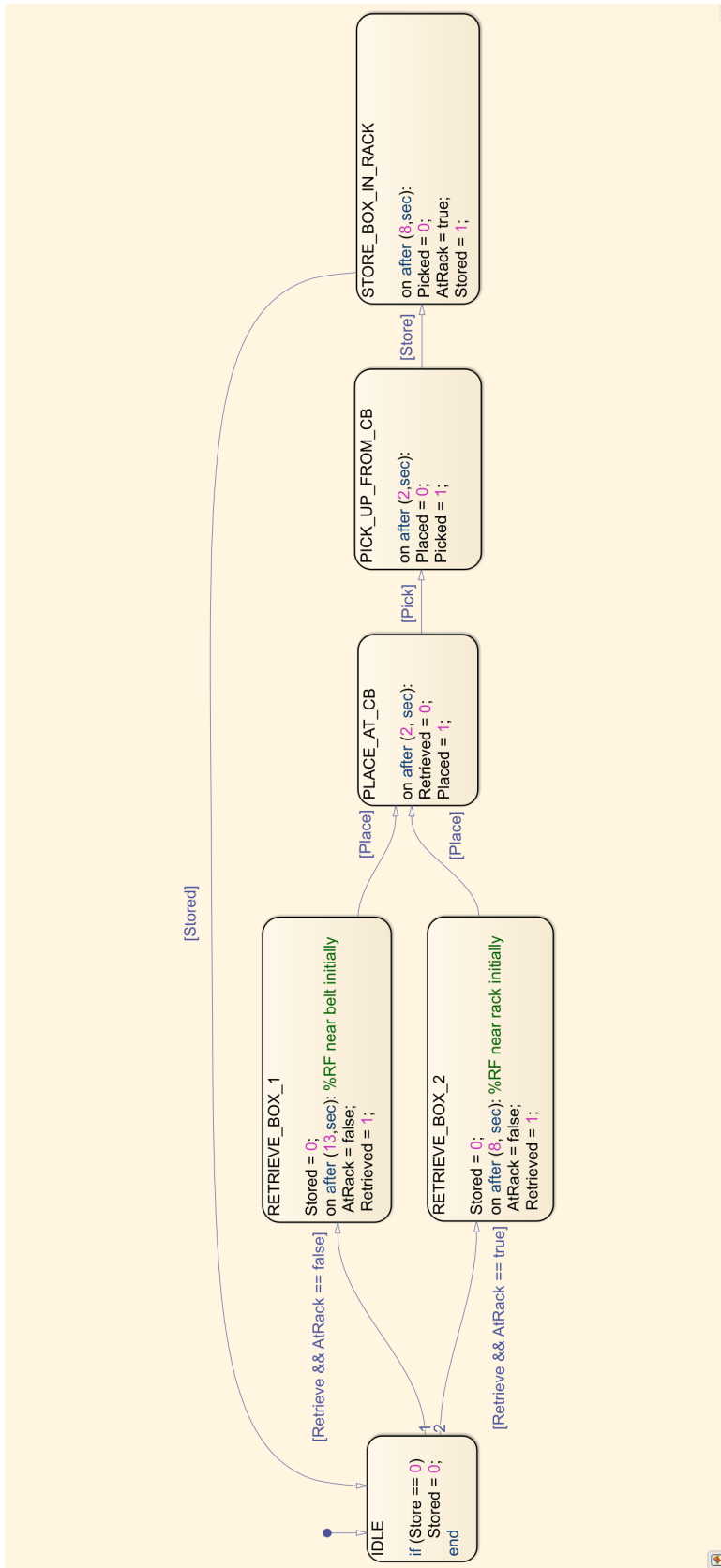


Figure D.7: Inside the rack feeder Stateflow chart.

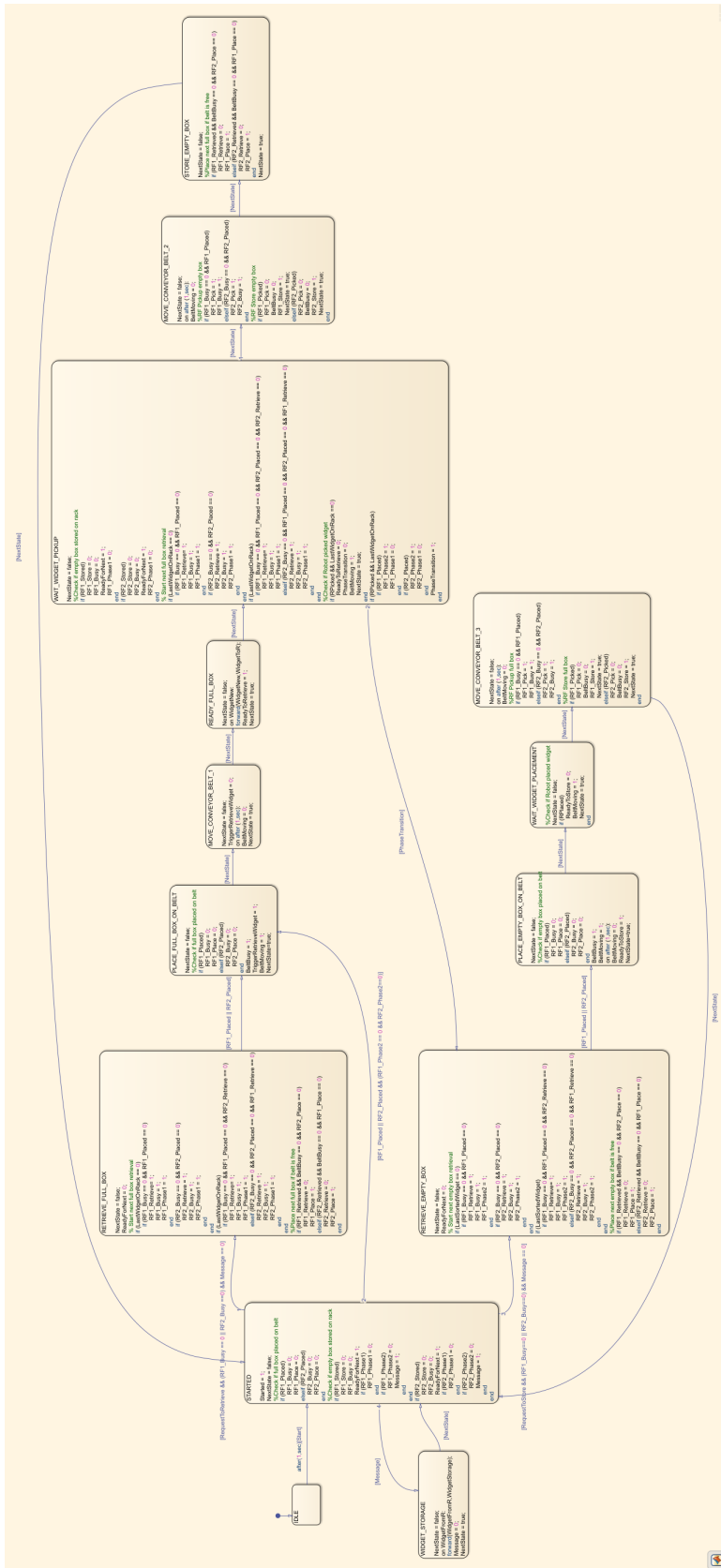


Figure D.8: Inside the Warehouse Stateflow chart (modified for two rack feeders).

Appendix E

MQTT in Simulink using MATLAB function block

E.1 Provisions in Warehouse subsystem

| Name | Scope | Port | DataType | InitialValue |
|-----------------------|--------|------|-------------|--------------|
| Start | Input | 1 | double | |
| RequestToRetrieve | Input | 2 | double | |
| WidgetNew | Input | 3 | Bus: Widget | |
| Picked | Input | 4 | double | |
| RequestToStore | Input | 5 | double | |
| WidgetFromR | Input | 6 | Bus: Widget | |
| Placed | Input | 7 | double | |
| NextState | Local | | boolean | false |
| RackFeederBusy | Local | | double | 0 |
| AtRack | Local | | boolean | false |
| BeltMoving | Local | | double | 0 |
| Started | Output | 1 | double | 0 |
| StartDigitalTwinRF | Output | 2 | double | 0 |
| TriggerRetrieveWidget | Output | 3 | double | 0 |
| ReadyToRetrieve | Output | 4 | double | 0 |
| WidgetToR | Output | 5 | Bus: Widget | |
| ReadyToStore | Output | 6 | double | 0 |
| WidgetStorage | Output | 7 | Bus: Widget | |
| ReadyForNext | Output | 8 | double | 0 |

Figure E.1: Information of input and output signals used in the Warehouse Stateflow chart of the benchmark model with modifications for facilitating MQTT (from Simulink Model Explorer).

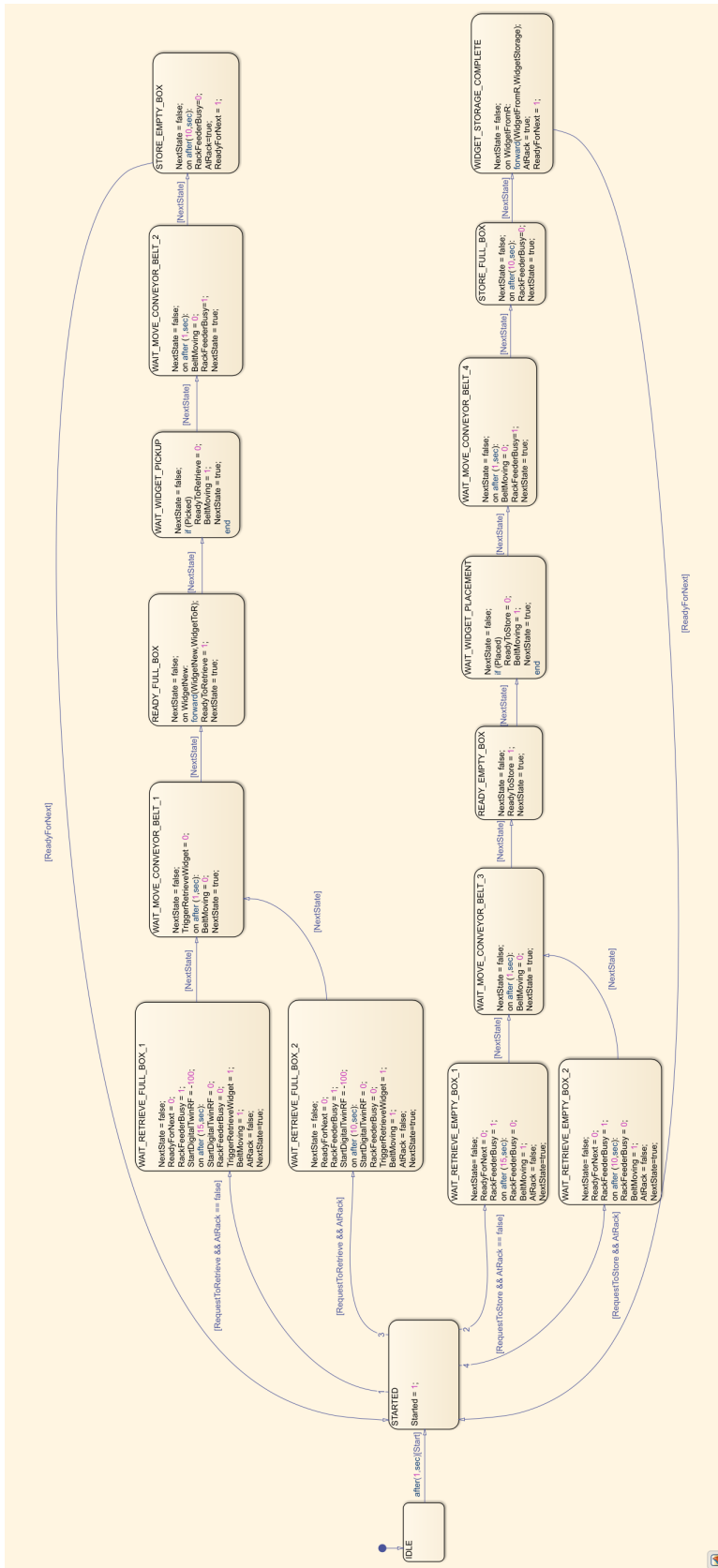


Figure E.2: Inside the Warehouse Stateflow chart of benchmark model (modified for facilitating MQTT).