Discrete event modelling, simulation and control of distributed manipulation environment

Naeem Anjum
University of Wollongong
DISCRETE EVENT MODELLING, SIMULATION AND CONTROL OF DISTRIBUTED MANIPULATION ENVIRONMENT

A thesis submitted in fulfilment of the requirements for the award of the degree of

Master of Engineering (Honours)

from

UNIVERSITY OF WOLLONGONG

by

Naeem Anjum

B.E.(Electronics Engg.), N. E. D University, Karachi, Pakistan, 1986

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING
March, 1995
DECLARATION

This is to certify that the work presented in this thesis was carried out by the author in the Department of Electrical and Computer Engineering, University of Wollongong, Australia and has not been submitted to any other university or institute.

..........................

(Naeem Anjum)
To my mother, aapi and the memory of my father
for the pains they suffered for me

To my wife
for the patience and support that she extended to
me throughout the course of this research
ACKNOWLEDGMENTS

The first word of gratitude goes to my project supervisor, Dr. Fazel Naghdy, for his superb guidance throughout this research work. Indeed, without his time and his invaluable suggestions, this project would not have been successful.

I would like to thank all of my laboratory colleagues and researchers for their cooperation and helpful discussions throughout my stay at the university. The cooperation and technical support of the departmental staff are also acknowledged.

I am also thankful to Ministry of Science & Technology, Govt. of Pakistan for providing me partial financial support to take up my studies at the University of Wollongong, Australia.
ABSTRACT

Conventional robotics has proved to be inflexible and non-generic and a radical change is required before robotics can be successfully used in automation of today's competitive industries. The primary aim of this project is to study the concept of distributed robotics in the context of Distributed Manipulation Environment (DME) as a new approach for flexible automation, and identify methodologies required for the design and development of such systems.

A DME system is inherently discrete and modular, and its operation can be specified as a chain of concurrent and sequential events. The overall DME model is the coupling of a set of component models. The component models are the abstractions of the real components used in DME, i.e., linear and rotary actuators, sensors, etc. These component models assist to create a distributed and hierarchically structured model of the system, identical to the actual system. They also simplify the task of development, debugging and maintenance of the whole system.

The work conducted in this thesis is mainly concerned with modelling, simulation and event-based control of DME. The modelling, conducted both at the atomic and the coupled level, is quite generic and provides a framework for static and dynamic behaviour analysis of DME systems. These models are then used to develop simulation models and event-based control of DME. The simulation models serve as a mean of performance evaluation of the system on a computer before the actual implementation in real-time, and thus play an important role in the design, modification and improvement of the DME systems. The event-based controller, employed for DME, provides a simple and robust control scheme. The controller, itself, can be tested, validated and finely tuned through simulation before implementation. It also provides comprehensive error messages for system diagnostic in case of any malfunctioning.
2.4.1 Queuing Network ................................................................. 18
2.4.2 Markov Chain Model ............................................................ 20
2.4.3 Petri Net Model ................................................................. 21
2.4.4 Automata & Finite State Machine Approach ......................... 23
2.4.5 Finitely Recursive Process Formalism .................................... 24
2.4.6 GSMP and Discrete Event Simulation Models ......................... 26
2.4.7 Min-Max Algebra Model ....................................................... 27
2.4.8 Temporal Logic Formalism ................................................... 28
2.5 Classification of DEDS ............................................................. 29
2.6 Applications of Discrete Event Modelling and Simulation in
Manufacturing and Automation .................................................... 30
  2.6.1 Short Term Planning System .................................................. 30
  2.6.2 Advanced Robot-Controlled Instrumentation for Fluid
          Handling Laboratory in Space ............................................... 31
  2.6.3 Manufacturing System Simulation Tool .................................. 32
2.7 Summary .................................................................................. 32

DISCRETE EVENT MODELLING OF DME 33

3.1 Introduction ............................................................................. 33
3.2 Discrete Event Modelling of DME ............................................. 33
  3.2.1 Atomic Model ................................................................. 34
  3.2.2 Coupled Model ............................................................... 36
3.3 DME System Model ................................................................. 38
3.4 DME Model Example ............................................................... 40
  3.4.1 Model of Drilling System ................................................... 42
  3.4.2 The Select Algorithm ......................................................... 44
3.5 Summary .................................................................................. 45
APPENDIX A

Code for Simulation of Drilling System ................................................................. 112

APPENDIX B

B.1 Code for Event-Based Controller of Drilling System ........................................ 119
B.2 Code for Control Signal of CM1 ........................................................................ 127

APPENDIX C

Code for EDS Simulation ....................................................................................... 136

APPENDIX D

D.1 Code for EDS Event-Based Controller ............................................................. 151
D.2 Code for Control Signal of CM1 ....................................................................... 166
D.3 Code for Control Signal of CM2 ....................................................................... 172
D.4 Code for Control Signal of CM21 ..................................................................... 189
D.5 Code for Control Signal of CM3 ....................................................................... 205
D.6 Code for Control Signal of CM4 ....................................................................... 210

APPENDIX E

Genetic Algorithm Based Job Optimization Code .................................................. 215

APPENDIX F

Event-Based Control Model of Drill Hole System

F.1 Event-Based Control Model for MM_B ............................................................. 227
F.2 Event-Based Control Model for CM1 ............................................................... 227
F.3 Event-Based Control Model for MM_C ............................................................. 227
F.4 Event-Based Control Model for MM_D ............................................................. 228

APPENDIX G

Event-Based Controller of EDS

G.1 Event-Based Control Model of CM1 ............................................................... 229
G.1.1 Event-Based Control Model of MM1..........................229
G.1.2 Event-Based Control Model of MM2..........................229
G.2 Event-Based Control Model of CM2............................230
  G.2.1 Event-Based Control Model of MM3..........................230
  G.2.2 Event-Based Control Model of MM4..........................231
  G.2.3 Event-Based Control Model for CM21.......................231
    G.2.3.1 Event-Based Control Model of MM5.....................231
    G.2.3.2 Event-Based Control Model of MM6.....................232
G.3 Event-Based Control Model of CM3............................232
  G.3.1 Event-Based Control Model of MM7..........................233
  G.3.2 Event-Based Control Model of MM8..........................233
G.4 Event-Based Control Model of CM4............................233
  G.4.1 Event-Based Control Model of MM9..........................234
  G.4.2 Event-Based Control Model of MM10..........................234
G.5 Event-Based Control Model of MMA............................234
LIST OF FIGURES

Figure 1.1: Manipulation Module; A Building Block of DME \[Nag93\] ......................... 4
Figure 1.2: Types of Manipulation Modules ................................................................. 5
Figure 2.1: Trajectory Behaviour of Continuous Time Models ...................................... 13
Figure 2.2: Trajectory Behaviour of Discrete Time Models ........................................... 14
Figure 2.3: Trajectory Behaviour of Discrete Event Models ......................................... 15
Figure 2.4: A Queuing Network ................................................................................... 18
Figure 2.5: A Markov Birth-Death Process .................................................................. 21
Figure 2.6: A Petri Net Model ...................................................................................... 22
Figure 2.7: A FSM Model ............................................................................................ 24
Figure 2.8: A GSMP Model ......................................................................................... 26
Figure 3.1: An Atomic Model ....................................................................................... 36
Figure 3.2: A Coupled Model ...................................................................................... 38
Figure 3.3: A Modular DME System ........................................................................... 39
Figure 3.4: DME Drilling System .................................................................................. 41
Figure 3.5: Manipulation Modules in Drilling Model .................................................... 42
Figure 4.1: The Control Module .................................................................................. 48
Figure 4.2: An Atomic Simulator .................................................................................. 52
Figure 4.3: A Coupled Simulator .................................................................................. 55
Figure 4.4: Evolution of Simulation Model by Processes ............................................. 58
Figure 4.5: Setting up of Parameters ............................................................................ 59
Figure 4.6: Initial Set Up .............................................................................................. 59
Figure 4.7: Marking Done ............................................................................................ 59
Figure 4.8: Mani. Modules Just Before 1st Hole .......................................................... 60
Figure 4.9: First Hole Drilled ...................................................................................... 60
Figure 4.10: Three Holes Drilled .................................................................................. 60
Figure 4.11: Job Completed .......................................................................................... 60
Figure 6.12: Control Signal of CM2 ................................................................. 99
Figure 6.13: Control Signal of CM21 ............................................................... 99
Figure 6.14: Control Signal of CM3 ................................................................. 99
Figure 6.15: Control Signal of CM4 ................................................................. 99
LIST OF TABLES

Table 2.1: Comparison of Three Types of Models ............................................................. 16
Table 2.2: Queue Parameters [Bro82] ................................................................................. 20
Table 2.3: Classification of DEDS Models [Ho89] ............................................................ 29
1.1 Problem Statement

The industrial robot was introduced in manufacturing industry as a flexible and intelligent tool for automatic manipulation. The experience gained over the last two decades has, however, shown that the employment of an industrial robot does not always produce a flexible automated system. An industrial robot is a flexible and programmable tool. The manipulation solutions offered based on an industrial robot are, however, mostly application oriented, non-generic and non-systematic. This has resulted in high complexity and high cost of systems developed based on industrial robots.

The solutions developed based on an articulated robot have normally proved unnecessarily complex to control, too difficult to integrate into a production line and too costly particularly for small to medium-size industries. Moreover, the approach has been radically different from the natural trends developed in the industry over the years and hence requiring specialised high-skilled personnel for its design, development and maintenance.

In the work conducted by Naghdy, a distributed approach to robotics has been proposed [Nag89], [Nag93]. The aim has been to define and develop a robotics system which is
more systematic, generic, flexible and economical than conventional systems. The theoretical basis of the proposed distributed robotics is the concept of Distributed Manipulation Environment (DME). DME is a distributed and concurrent system formed by a network of Manipulation Modules that are the entities capable of mechanical, informational, sensory and processing behaviour.

In order to control a DME to achieve a desired performance, its behaviour should be formally defined and modelled. Considering the nature of DME, it is possible to identify two different levels of modelling:

(a) The component level at which the behaviour of the individual manipulation modules working together is analysed and modelled.

(b) The functional level at which the interaction, coordination and sequence of the operation of the manipulation modules should be formally and systematically defined and modelled.

At level (a) the conventional analytical methods are used to describe the kinematics and dynamics of the manipulation modules. The modelling procedure for level (b) is fundamentally different from (a) and new tools and techniques are required. Study of the modelling of DME at this level, computer simulation of the model and its control system are the main issues addressed in this work.

1.2 Thesis Aims and Objectives

This research complements the work conducted previously on DME that mainly concentrated on the design and development of a distributed and concurrent computing platform for it [Tou93]. The primary aim of this work is to study the modelling, simulation and control of DME at the functional level.

Job scheduling in DME is also addressed briefly in this work. A job optimizing algorithm based on genetic algorithm aiming at reducing the overall manipulation time of a DME system has been developed.
A DME system is basically a Discrete Event Dynamic System whose operation can be specified as a chain of concurrent and sequential events. At present no general methodology exists to describe the dynamic behaviour of such a system. Researchers in the past have applied deterministic techniques such as Min-Max Algebra [Ina90], Finitely Recursive Process [Coh85] and stochastic approaches like Markov Chains [Tah92], Queuing Networks [Bro82] to model the discrete event dynamic systems.

In this work Discrete Event System Specifications formalism, proposed by Ziegler, is employed [Zie76], [Zie84]. The research work has focussed to achieve the following objectives:

i) To study DME as a discrete event dynamic system and develop a discrete event model for it.

ii) To describe a discrete event simulation model for DME.

iii) To develop an event-based control model for DME.

iv) To develop a job optimizing algorithm based on genetic algorithm for DME.

The simulation models are implemented using Simscript which is a discrete event simulation package.

The Discrete Event System Specifications formalism does not only capture the dynamics and concurrent behaviour of the system but also provides a formal basis for specifying the dynamic model within the discrete event simulation environment. This formalism also facilitates an event-based control paradigm for the control of a DME system.

1.3 Distributed Manipulation Environment

Distributed Manipulation Environment (DME) proposes a distributed approach to robotics and flexible automation [Nag89], [Tou91]. In the context of this work a distributed manipulation environment is an inherently distributed, intelligent, and
programmable system that has been hardened in mechanics and control software to perform a specific type of manipulation. The basic unit of DME is a manipulation module which is a stand-alone, autonomous and intelligent unit. A manipulation module is capable of mechanical, informational, sensory and processing behaviour and is loosely coupled to other manipulation modules in the system. Each manipulation module has mechanical links for connection to other manipulation modules, electronics links to communicate with them and sensory links to sense and realise its environment as shown in figure 1.1.

![Diagram of Manipulation Module](image)

Figure 1.1: Manipulation Module; A Building Block of DME [Nag93]

There are different types of manipulation modules defined to perform different tasks. These manipulation modules, based on the type of actuation they perform, can be categorised as shown in Figure 1.2.

An ensemble of manipulation modules and standard/non-standard hard automation components produce a Distributed Manipulation Environment which is configured and linked (hardened) to perform a specific task. Each manipulation module can be programmed individually. All these modules communicate locally and coordinate their operations on the basis of information received from their neighbours and a priori knowledge on the required task. Such a coordination is vital to avoid conflict and undue
competition for available resources. The efficiency of the whole system depends on this coordination rather than the capabilities of the individual manipulation modules.

The distributed approach employed in DME not only simplifies the overall design and development of the system but is also independent of the number of manipulation modules used in the system. This number can vary from one system to another, depending on the functionality of the system. An existing DME can be easily modified to perform a different task by adding or removing manipulation modules and changing their configurations.

![Types of Manipulation Modules](image)

Figure 1.2: Types of Manipulation Modules

The result is a highly modular and flexible design, reconfigurable at minimum effort and cost. This is in contrast to the conventional robotics where radical modifications in the overall system are required to accommodate any change in the task being carried out by the system.
1.4 Discrete Event System Specifications Formalism

Discrete event system specifications (DEVS) formalism was proposed by Ziegler in 1976 [Zie76]. The DEVS formalism provides a formal basis for specifying discrete events dynamic systems that are amenable to mathematical manipulation for their behaviour analysis.

The DEVS is a structure

\[ < U, Y, S, \delta, \lambda, ta > \]

where

- \( U \) is the input set
- \( Y \) is the output set
- \( S \) is the state set
- \( \delta \) is the transition function
- \( \lambda : S \rightarrow Y \) is the output function
- \( ta : S \rightarrow \mathbb{R}^+_{0,\infty} \) is the time advance function where \( \mathbb{R}^+_{0,\infty} \) is the set of non-negative reals with \( \infty \) adjoined.

The transition function \( \delta \) can further be divided as follows [Zie84].

- \( \delta_{\text{int}} : S \rightarrow S \) is the internal transition function and
- \( \delta_{\text{ext}} : Q \times U \rightarrow S \) is the external transition function that is applied to the events in the input signals with \( Q \) defined as

\[ Q = \{ (s, e) \mid s \in S, 0 \leq e \leq t_a(s) \} \]

The above DEVS model, like a system theoretical model, may be considered as a black box that contains a process and produces outputs in response to inputs. A DEVS model, however, differs from the classical system theory in its inclusion and emphasis on the concept of an event [Fis91]. An event can be either internal or external. An internal event is defined by \( \delta_{\text{int}} \) and occurs when some conditions of its occurrence are fulfilled. The external events are the input stimuli and are modelled by \( \delta_{\text{ext}} \).
Furthermore, time in DEVS model does not increase continuously or in fixed steps. It progresses from one event to another so that the system time gets incremented in chunks of variable size.

### 1.5 Simscript

Simscript II.5 has been used in this work for discrete event simulation of the developed models. Simscript II.5, with its integrated graphic interface, Simgraphic, is a free form, English-like and general purpose programming language [Gar90], [Sim91], [Rus91]. It has been specifically designed for simulation purposes and allows modular implementation.

A simulation model in Simscript II.5 is built around simscript objects that include temporary entities\(^1\), permanent entities, processes and resources. An entity may have a number of its instances in the model and may also be assigned with a number of attributes. All the instances of a particular entity, though have the same name, yet have their own values for the attributes. These entities may also be organised into sets to serve as an ordered list with various ordering disciplines. They are declared in the preamble.

A process expresses the dynamic behaviour of a Simscript model. It describes an object and the sequence of its interrelated events separated by the lapses of time. It can also activate, interrupt, suspend or resume another process in the system.

Resources are the passive elements of a model and are used to model objects required by the process objects. Processes may request or relinquish resources, automatically waiting for those which are unavailable when requested and automatically starting other processes by relinquishing unneeded resources.

---

\(^1\) All italic words are part of Simscript II.5 vocabulary
Simscript II.5 has also a full range of input/output capabilities and allows an easy incorporation of presentation graphics and animation in the program. The presentation graphics and icons are usually created by the graphic editor and may be modified later on without having to modify the program.

1.5.1 Simscript Simulation Model

A simscript simulation model consists of three primary elements of [Rus90], [Sim91a]:

- preamble
- main
- process routine for each process declared in the preamble

The preamble sets up the static structure of the model. The static parameters of the DEVS model are, therefore, declared in preamble. It has no executable command and has the following structure.

\textit{preamble.sim}

-----------------------------------------------
preamble
process declaration
resources declaration
entities declaration
global variables declaration
end

-----------------------------------------------

The main is the module where execution of the simulation model begins and it is done by activating the initial processes. The main module has the following structure.

\textit{main.sim}

-----------------------------------------------

main
local variable declaration, if any
initialization of resources
necessary executable command
start simulation
end

A process routine captures the dynamics of the object that it is representing. In this work it thus contains code on how the transition function of the DEVS model is implemented. Its structure is as under.

process\_name.sim

process name
local variable declarations that are particular to this process
code for dynamic behaviour of the object
end

1.6 Genetic Algorithms

Genetic algorithm, proposed by J. Holland in 1975, is a multiple-point search and randomised optimization technique for functions defined over finite (discrete) domain [Gol89]. Based on the principles of biological evolution and natural selection found in plants and animals, genetic algorithms (GAs) provide a superior approach to global optimization that differs considerably from heuristic methods and approaches like simulated annealing ([Kir83], [Dav87]) and optimizing by artificial neural networks [Hop85].

GAs work from a population (data group) instead of a single point. They start by operating on an initial population through a set of stochastic state transition genetic
operators that generate a new population, more biased towards the most promising area of the search space. The initial population can be chosen heuristically or at random. This process is called reproduction. The reproduction process continues to repeat itself and the population improves from generation to generation until some of the "off-springs" (members of the population also called chromosomes) reach the objective function or are at a stage where no further improvement can be made. Unlike other methods, GAs tend to move the entire population toward the best solution thus greatly reducing the probability of being stuck in local maxima [Hor92].

1.7 Structure of the Thesis

The thesis comprises of seven chapters. The goals and objectives of the study are given in chapter 1. In addition, a brief introduction to Distributed Manipulation Environment, DEVS formalism, Simscript and Genetic Algorithm which form the significant concepts employed in this work are provided in this chapter. The structure of the thesis is also introduced in chapter 1.

The background of the project is studied in chapter 2. Various aspects of the modelling techniques, used for discrete event dynamic systems (DEDS), including their merits, applications and limitations are critically reviewed. A number of real-time applications of discrete event modelling and simulation have also been discussed briefly.

The discrete event modelling of Distributed Manipulation Environment using DEVS formalism is reported in chapter 3. The modelling has been carried out at both atomic and coupled levels. This allows modular and hierarchical construction of discrete event models of DME systems.

The discrete event simulation environment for DME is described in chapter 4. The mathematical discrete event models of DME that are developed in chapter 3 and are used as a basis to develop discrete event simulation models. The simulation models are
implemented using Simscript and Simgraphic. The implementation procedure is also explained in this chapter.

Event-based control logic and control models required for DME are studied in chapter 5. The event-based model of DME is derived from the mathematical discrete event model developed in chapter 3. The control models are also simulated using Simscript. These models can be used in real-time DME system to control its operation. The optimization of job scheduling in DME has also been addressed in this chapter. Genetic algorithms have been used for this purpose.

The validation of the developed methodologies is presented in chapter 6. A hypothetical production line for drilling steel plates is modelled. Its operation and control based on the developed models is then simulated on Simscript. The feasibility of the approach is demonstrated in this study.

The critical study of the findings of the project is carried out in chapter 7. During this study some conclusions on the work are drawn and suggestions for further work are also offered.
2.1 Introduction

In this chapter a study of the modelling approaches particularly with respect to Discrete Event Dynamic Systems (DEDS) has been carried out. This can serve as a basis for better understanding of the work described in the chapters to follow. The features of DEDS and different modelling approaches used are described briefly, and their applications, merits and limitations are highlighted. A comparison of these modelling approaches, based on their constructs and the time-base they employ, has also been carried out. A brief description of three real-time applications of discrete event modelling and simulation in manufacturing and robotics has also been provided.

2.2 Modelling and Type of Models

The use of models is essential for a better understanding of a problem, system or phenomenon and developing a solution for it. Modelling provides a logical, systematic, efficient and explicit way for analysis and judgement. It is, however, of fundamental importance that the model should be a valid representation of the system so that it poses the same problem and behaviour characteristics as that of the system being studied. This
will ensure identical behaviour for both the model and the system under different varying conditions, and the results obtained from the performance of the model would be correct and applicable to the real world system [Sha75].

There are basically three types of models for dynamic system specification [Del89].

- Continuous Time Model
- Discrete Time Model
- Discrete Event Model

### 2.2.1 Continuous Time Models

Continuous Variables Dynamic Systems (CVDS) are expressed by continuous time models. A CVDS is the natural dynamic system. It evolves gradually and continuously over time such that in a finite time span, the state variables often change their values infinitely [Cel91]. As a result, the state trajectory of a CVDS is continuously changing as the states take values in $\mathbb{R}^n$ and are driven by continuous inputs. A typical CVDS trajectory is shown in figure 2.1.

![Figure 2.1: Trajectory Behaviour of Continuous Time Models](image)

There are various approaches available for modelling the dynamic behaviour of a CVDS. They mainly capitalize upon empirical techniques, first principals, input/output data or heuristic information for deriving a set of differential equations, statistical response-
surface models or some suitable transfer functions that faithfully describe their state trajectories [Kam94].

The continuous time models are further divided into lumped parameter models and distributed parameter models. The lumped parameter models are characterised by ordinary differential equations of the general form

\[ x'(t) = f(x, u, p, t) \]

where

- \( x \) is the state vector
- \( u \) is the input
- \( p \) is a set of parameters
- \( t \) is the time

The distributed parameter models are described by partial differential equations which contain derivatives with respect to different variables such as space and time coordinates.

### 2.2.2 Discrete Time Models

Discrete time models are used whenever the state of a system depends on one or more variables that can only assume a discrete set of possible values. In discrete time models, the time axis is not continuous as is in CVDS. Rather, as shown in Figure 2.2, it is discretized.

![Figure 2.2: Trajectory Behaviour of Discrete Time Models](image-url)
The time is advanced by a fixed increment $\Delta t$, sufficiently large to make it noteworthy [Gre73]. As a result, the discretization is equally spaced by $\Delta t$.

The system is modelled by a set of difference equations of the form
\[ x_{k+1} = f(x_k, u_k, t_k) \]
where
- $x_{k+1}$ is the next state in which the system will be after time $\Delta t$, being initially at state $x_k$
- $u_k$ is the input at time $t_k$
- $t_k$ is the time corresponding to system state $x_k$

### 2.2.3 Discrete Event Models

Discrete event models are used to model discrete event dynamic systems (DEDS). DEDS are essentially man-made systems and their characteristics differ considerably from those of CVDS, the natural dynamic systems. Consequently, DEDS cannot be modelled by ordinary or partial differential equations.

A DEDS evolves in time by the interactions of the timings of different events so that in a finite time span, only a finite number of state changes may occur. Figure 2.3 shows a typical state trajectory, which is piecewise constant and event driven.

![Figure 2.3: Trajectory Behaviour of Discrete Event Models](image)

Figure 2.3: Trajectory Behaviour of Discrete Event Models
Each constant segment in the state trajectory represents a different system state. The lengths of these segments correspond to the time for which the system remains in the states indicated by that segments. This state and holding time pair characterises the state trajectory of the system [Khe88].

Each event in DEDS occurs at a specific time and affects only a part of the variables that describes the state of the system while leaving the others unchanged. Furthermore, these events occur abruptly and at times that are often irregular and unknown.

### 2.2.4 Comparison of the Models

The three types of models discussed above have been compared on the basis of their time-base, input segments, state trajectory and output. Table 2.1 shows the result of this comparison.

<table>
<thead>
<tr>
<th>Table 2.1: Comparison of Three Types of Models</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Continuous Time Model</strong></td>
</tr>
<tr>
<td>Input</td>
</tr>
<tr>
<td>Time-Base</td>
</tr>
<tr>
<td>Output and State Trajectories</td>
</tr>
</tbody>
</table>

### 2.3 Discrete Event Modelling and Simulation

Discrete event modelling and simulation play a fundamental and important role in understanding and developing complex discrete event dynamic systems (DEDS). Such systems can be found today in a wide variety of technological areas such as flexible
manufacturing, assembly and production lines, traffic systems, computer and communication networks, etc [Den89].

Since a DEDS is a man-made system rather than a natural/physical one, no physical law exists for DEDS, as they are for CVDS. The natural limits of materials and ergonomics are then the only factors that may constrain a DEDS system configuration. System complexity can, therefore, easily explode in a combinational fashion making performance analysis and optimization a difficult job [Ho89].

The existing analytical approaches developed for performance evaluation of a system are often insufficient and inapplicable to complex discrete event dynamic systems. These methods are based on unrealistic assumptions and many simplifying approximations. The resulting inaccuracies, owing to these assumptions and approximations, make the result unacceptable for complex DEDS of practical interest. In such a situation discrete event modelling and simulation become the most important tools for [Rig89]:

- Understanding the behaviour of the system
- Testing the operation of the system during development
- Getting estimates of the performance measures of the system
- Making improvement and modification in the system before it actually goes into service

Such analysis and study of a system may otherwise be too costly, impractical or even impossible to be conducted directly on the system.

### 2.4 DEDS Modelling Approaches

A number of different modelling approaches have been proposed for analysing DEDS. Different DEDS models concentrate on different aspects of a system and are suitable for different purposes. Among these models are:

- Queuing Network Model
Chapter 2  

Background

- Markov Chain Model
- Petri Net Model
- Automata and Finite State Machine
- Finitely Recursive Process
- Generalised Semi-Markov Process and Discrete Event Simulation Model
- Min-Max Algebra Model
- Temporal Logic Model

### 2.4.1 Queuing Network

A queuing network models a DEDS in terms of a set of customers, a set of servers and an order in which the customers arrive and are processed [Dra87]. It can be viewed as a birth-death process having a population of customers that are either currently in service or waiting for being served. A birth is the arrival of a customer at the service facility while a death occurs when a customer departs from the facility. The state of the system is the number of customers in the facility. Figure 2.4 shows a queuing network for a simple DEDS.

![Figure 2.4: A Queuing Network](image)

A queuing system is characterised by five components [Tah92]:

- The arrival pattern of customers
- The service pattern
- The number of servers
• The capacity of the facility to hold customers
• The queue discipline

Both the arrival pattern and the service pattern are normally defined in terms of inter-arrival time and service time respectively. These times may either be deterministic or random with a specified probability distribution.

The system capacity refers to the maximum number of customers. It includes both those in services and those in queues waiting for their turns for the service. If a customer comes at a facility that is full, the arriving customer is not attended and forced to leave the facility without receiving service. The queue discipline is the order in which customers are served at the facility. It can be on a first-in, first-out (FIFO) basis, last-in, first-out (LIFO) basis, a random basis or a priority basis. These queuing system characteristics are generally described by Kendall's notion as [Bro82]

\[ v/w/x/y/z \]

where

• \( v \) indicates arrival pattern
• \( w \) indicates service pattern
• \( x \) indicates number of available servers
• \( y \) indicates system capacity and
• \( z \) indicates queue discipline

Various notations used for the three of these characteristics are listed in table 2.2. If \( y \) or \( z \) is not specified, it is taken to be \( \infty \) or FIFO respectively.
### Queue Characteristics

<table>
<thead>
<tr>
<th>Inter-Arrival Time or Service Time</th>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>(D)</td>
<td>Deterministic</td>
</tr>
<tr>
<td>M</td>
<td>(M)</td>
<td>Exponentially Distributed</td>
</tr>
<tr>
<td>(E_K)</td>
<td>(E_K)</td>
<td>Erlang-Type-K Distributed ((k = 1, 2, 3, \ldots))</td>
</tr>
<tr>
<td>G</td>
<td>(G)</td>
<td>Any Other Distribution</td>
</tr>
</tbody>
</table>

**Queue Discipline**

<table>
<thead>
<tr>
<th></th>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIFO</td>
<td></td>
<td>First-in, First-out</td>
</tr>
<tr>
<td>LIFO</td>
<td></td>
<td>Last-in, First-out</td>
</tr>
<tr>
<td>SIRO</td>
<td></td>
<td>Service in Random Order</td>
</tr>
<tr>
<td>PRI</td>
<td></td>
<td>Priority Ordering</td>
</tr>
<tr>
<td>GD</td>
<td></td>
<td>Any Other Specialized Ordering</td>
</tr>
</tbody>
</table>

**Table 2.2: Queue Parameters [Bro82]**

### 2.4.2 Markov Chain Model

The Markov chain model is used to study the short and long term behaviour of stochastic DEDS [Rab89]. A Markov process is a stochastic process in which the occurrence of the next state depends only on the current state and is independent of all the previous states through which the system has reached the current state. Formally a family of random variables \(\{q_{tn}\}\) constitutes a Markovian process if [Tah92]

\[
P \{ q_{tn} = x_n \mid q_{tn-1} = x_{n-1}, \ldots, q_{t_0} = x_0 \} = P \{ q_{tn} = x_n \mid q_{tn-1} = x_{n-1} \}
\]

for all possible values of \(q_{t_0}, q_{t_1}, \ldots, q_{tn}\). The above expression represents the conditional probability of the system for being in state \(x_n\) at time \(t_n\), given that it was in state \(x_{n-1}\) at time \(t_{n-1}\).

As shown in figure 2.5, a Markov process can be used to represent a stochastic DEDS as a birth-death process in which transition probabilities for moving from one state to another depends on the immediately preceding state only.
The markov chain models have their limitations. They require that all time parameters be exponentially distributed. Besides, in most practical problems, the number of states tends to be very large [Ibe93].

The markov chain model differs considerably from the queuing network. In markov chain model, though the times for the new birth and death are generated afresh in each state, only one of them is used to determine the next state and the other is discarded. In queuing network, the unused parameter is not discarded and is used to determine the event times in the next state. However, owing to the memory-less property of the exponential distribution, these modelling methods are statistically indistinguishable though algorithmic implementations of these models are, in general, different [Cao90].

2.4.3 Petri Net Model

Petri nets are used to model systems that are characterised as being concurrent, asynchronous, parallel deterministic and/or stochastic [Joh92], [Mur89]. A petri net is a directed graph \( G = (V, E) \) that satisfies:

\[
V = P \cup T \quad \text{and} \quad \text{where}
\]

- \( V \) is the set of vertices or nodes of \( G \)
• $P = \{ p_1, p_2, \ldots, p_n \}$; a set of places
• $T = \{ t_1, t_2, \ldots, t_m \}$; a set of transitions
• $E$ is the set of edges or arcs of $G$; a mapping from $P$ to $T$ or $T$ to $P$

In petri nets, places represent conditions, transitions represent events and the dynamic behaviour of the system is expressed by token. Typically, places are drawn as circles, transitions as bars or rectangular boxes and tokens as black dots, as shown in figure 2.6.

A place may either be an input to a transition or an output of a transition. Tokens are assigned to places and the presence of at least one token in a place indicates that the condition for the occurrence of the transition has been met. Such a place is called a marked place and the petri net is then called as marked petri net. The marking of a petri net corresponds to a state of the system being modelled.

The petri net moves to another system state by firing transitions. A transition can fire only if each of its input places contains at least one token. The occurrence of a transition removes one token from each input place and one token is added to each output place. This way the marking of a petri net will keep on changing with each transition that corresponds to the occurrence of an event. These firing of transitions and the resulting changes in marking in petri net represent the evolution of the system through different states.

Figure 2.6: A Petri Net Model
An extension of the above standard petri net is the timed petri net model that incorporates the system timing in it [Pet86]. To introduce this notion of time, each transition is associated with a firing delay. This firing delay is the time between enabling and firing of a transition and may either be deterministic or random with a specified distribution. Graphically an immediate transition (firing delay = 0) is indicated by a solid bar while a timed transition is represented by a hollow bar with the delay time written next to it.

Petri nets have a good descriptive power but have a disadvantage of greater language complexity.

### 2.4.4 Automata & Finite State Machine Approach

A Finite State Machine (FSM) model for DEDS is specified as a five tuple as [Ram89]

\[
FSM = \{Q, \Sigma, \delta, q_0, Q_m\}
\]

where

- \(Q\) is a finite set of states of system
- \(\Sigma\) is a set of all event labels
- \(\delta : Q \times \Sigma \rightarrow Q\); the transition function
- \(q_0 \in Q\) is the initial state
- \(Q_m \subseteq Q\) is the set of marker states

The transitions between states are triggered by events and are enabled or disabled by control variables. The sequence of events controls the evolution of the system in time as shown in figure 2.7.

The FSM approach models a DEDS as a generator of a formal language [You91]. The language generated by the generator must belong to some specification languages. A language in FSM is a set of event strings from \(\Sigma^*\) (\(\Sigma^* = \Sigma \cup \emptyset\)) and represents the desired behaviour of the system.
The primary goal of FSM is to design a controller. The controller obtains information during the system evolution and then based on this information, enables or disables control variables so that only legally admissible state transitions can take place. The formalism is not intended for the performance evolution of a DEDS. Rather, the aim is to construct a set of formal mathematical machinery to state precisely different qualitative properties of DEDS.

\[
\Sigma = \{ \alpha, \beta, \gamma, \mu, \lambda \}
\]
\[
X = \{ 0, 1, 2, 3, 4 \}
\]

Figure 2.7: A FSM Model

The FSM is a comparatively simple DEDS modelling approach but suffers from an exponential explosion in the number of its states.

### 2.4.5 Finitely Recursive Process Formalism

Finitely Recursive Process (FRP) is an algebraic approach to model DEDS. It describes a DEDS in terms of a set of functions that are the operations defined on processes [Ina88], [Coh85]. A process is a triple:

\[ \{ \text{tr } P, \alpha P, \gamma P \} \]

where

- \( \text{tr } P \subseteq \Sigma^* \); it specifies the traces (a sequence of discrete events) that \( P \) can execute.
• $\Sigma^*$ be the set of all possible finite sequences of events in $\Sigma$, the set of all events, including empty trace $\phi$.

• $\alpha P$ : event function that describes the trace that $P$ may execute or block after the execution of the current trace.

• $\gamma P : tr P \rightarrow \{0,1\}$; termination function that indicates whether $P$ will terminate or continue after executing the current trace.

These functions operate on processes to combine, concatenate and modify them according to various series, parallel, enabling and disabling operations of one or more DEDS in the system.

Formally, a process $Y$ is a finitely recursive process if it is expressible in the form

$$X = F (X)$$

$$Y = G (X)$$

where

• $F$ & $G$ are the functions on $\Pi$, the space of all processes and

• $X$ is a process

Thus FRP $Y$ has a finite recursive representation analogous to difference equation [Spa91]

$$x(t+1) = f (x(t)),$$  $$y(t) = g (x(t))$$

$$t = 0,1,2,........$$

for control systems.

As almost all DEDS perform some repeatable task; the processes associated with a DEDS can be represented as finitely recursive processes. It then becomes possible for a DEDS to determine its event strings recursively.

FRP approach for DEDS is more powerful then petri nets and finite state machines in the sense of set of event string that it generates. Also FRP, unlike FSM, does not suffer from an exponential increase in the number of its states with the increasing size of a DEDS, though is more complicated [Cao90].
2.4.6 GSMP and Discrete Event Simulation Models

The Generalised Semi-Markov Process (GSMP) model focuses on the formalisation of a discrete event simulation language for describing discrete event dynamic systems and also provides a mathematical environment within which DEDS can be analysed [Gly89], [Ho87].

Let $S$ & $E$ be the states set and events set of a DEDS respectively. For each $s \in S$, there is a list of events $E(s)$; a non-empty and finite subset of $E$; that contains those events that may occur when the system is in state $s$, as shown in figure 2.8.

![Figure 2.8: A GSMP Model](image)

Among all members of $E(s)$, however only one event can occur to trigger the system into the next state. Each event in $E(s)$, is assigned with a clock, the reading $c_e$ of which indicates the amount of time that has passed since the clock was last activated. All the events in $E(s)$ compete with each other to cause a transition out of state $s$. To select an event to occur, event life times of all events in $E(s)$ are determined according to a given distribution. The event life time of an event is the time after which the event is going to trigger the transition out of the current state provided no other event occurs before this event. The event $e \in E(s)$ that actually occurs is the one that has the minimum event life time.
Upon occurrence of this event, the system instantaneously transit to a new state determined by the arbitrary logical conditions enabled at the time of transition together with appropriate distribution function [Ho87]. A new event list and event life times corresponding to the new state are generated. The process then repeats and this way the system evolves in time.

### 2.4.7 Min-Max Algebra Model

Min-max algebra approach captures the transient behaviour of a system and is mainly used to model deterministic DEDS, though some stochastic extensions of it are now being proposed. In min-max algebra, system variables interact with each other through two operations; maximisation and addition [Inr90]. These operations are denoted by max and $\oplus$ and are defined as

- **Product:** $a \times b = a \oplus b$
- **Addition:** $a + b = \max(a, b)$

As mentioned in section 2.4.3, a petri net is used to model a system in which various components of the system work concurrently. A simple petri net in which there is a single transition upstream, and a single transition down stream in every place is called an event graph. Since in petri nets a transition can take place only if each of its input places contains at least one token, event graphs cannot model logical OR condition [Coh89]. Thus the system state equation of a non-concurrent DEDS, if modelled by event graphs, comes out to be non-linear. The same system, however, can be modelled as a linear system if the algebra employed for the system state equations is the min-max algebra [Ols93]. The min-max algebra, in fact transforms the well-known properties within conventional algebra into the theory of DEDS.
2.4.8 Temporal Logic Formalism

Temporal logic is a logic of propositions whose falsity or truth may depend on time. This logic is an extension of the classical logic to include the notion of time. A Temporal Logic Model (TLM) for a DEDS is defined as [Lin92], [Kro87]

\[ TLM = (V, F^*, S_0, l) \]

where

- \( V = (S, E, f) \); an event structure in which
  - \( S \) is the set of states
  - \( E \) is the set of events
  - \( f : E \times S \rightarrow S \) such that \( \forall s \in S, \forall e \in E(s), f(e, s) \) is defined
  - \( E(s) \) is the set of events which are firing in the state \( S \)
  - \( F^* \) is the set of all subsets of \( F \)
  - \( F \) is the set of temporal logic formulae
  - \( S_0 \) is the initial state
  - \( l : E \times S \rightarrow F^* \); a labelling function which associates to every pair \((e, s)\) the set of formulae that hold in \((e, s)\)

The temporal logic formulae are obtained by applying temporal operators to system state formulae. The five basic temporal operators are [Bun73]:

- \( O \) next operator
- \( \square \) always or henceforth operator
- \( \diamond \) eventually or sometime operator
- \( U \) until operator
- \( P \) precedes operator

In TLM, a set of rule based on temporal logic formulae is constructed that ensures a correct DEDS behaviour and imposes restrictions so that illegal states cannot be reached during system evolution. The set of these rules is called supervisory set of rules.
2.5 Classification of DEDS

The above discussed DEDS models can be classified in many ways. On the basis of time, they can be categorised into timed and untimed DEDS.

Timed DEDS contain time as an integral part of the model and are used when performance evaluation of the system, e.g., system throughput, mean sojourn time, etc., is the main concern.

Untimed DEDS, on the other hand, emphasis on the state or event sequence of the system and are more suited for logical or qualitative behaviour analysis.

On the basis of their construct, DEDS models can be divided into 3 groups of logical, algebraic and performance DEDS. Table 2.3 shows the result of these classifications of DEDS.

<table>
<thead>
<tr>
<th>DEDS</th>
<th>TIMED</th>
<th>UNTIMED</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOGICAL</td>
<td>• Temporal Logic</td>
<td>• Finite State Machines</td>
</tr>
<tr>
<td></td>
<td>• Timed Petri Nets</td>
<td>• Petri Nets</td>
</tr>
<tr>
<td>ALGEBRAIC</td>
<td>• Min-Max Algebra</td>
<td>• Finitely Recursive Process</td>
</tr>
<tr>
<td>PERFORMANCE</td>
<td>• Markov Chains</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>• Queuing Networks</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• GSMP/Simulation</td>
<td></td>
</tr>
</tbody>
</table>

Table 2.3: Classification of DEDS Models [Ho89]


2.6 Applications of Discrete Event Modelling and Simulation in Manufacturing and Automation

Most of the manufacturing and automation applications have a DEDS structure at some level of description [Özv90]. Discrete event modelling and simulation thus may be applied to these systems not only at the design stage but also in handling of tasks during daily routine operation. Some applications of discrete event modelling and simulation have already been successfully implemented in industry while others are finding their way very rapidly. A brief description of three applications of discrete event modelling and simulation in real time is given below.

2.6.1 Short Term Planning System

The Short Term Planning (S-Plan) system is a discrete event modelling and simulation based system that was recently developed for the UK paper and board industry [Bry89]. This system performs the function of factory wide control that includes the integration, control and management of the whole of the business and production process. It is being currently used in Stoneywood mill at Wiggins Teape. The Stoneywood mill is one of the largest and most complex of the UK paper mills and has a manufacturing capacity of 70,000 tonnes per annum of high value added paper.

The S-Plan at stoneywood controls whole of the manufacturing process on a 24 hour a day basis. It translates orders into programmes of work for each of its 40 production centres. Generally orders for a period of six weeks are planned. There might be as many as 1000 orders being processed on the shop floor. The system also accepts feedback dynamically from the shop floor concerning the status of all the operations and updates itself rapidly in the event of any break down, change of resources or sudden change in the production pattern. It is also capable of providing information about bottle necks, slack period in the current production plan, and the effects of overtime, sub-contracting
and new orders, etc., to management. The benefits obtained by using S-Plan include [Gil89]:

- Finishing productivity up by 25%
- Site output up by 20% and
- Customer complaints reduced from 27/1000 to 9/1000 tonnes.

### 2.6.2 Advanced Robot-Controlled Instrumentation for Fluid Handling Laboratory in Space

The second application of Discrete event modelling and simulation discussed here is about the advanced robot-controlled instrumentation for Fluid Handling Laboratory (FHL) in a semi autonomous environment [Sar90]. FHL is part of Life Saving Module (LSM) of NASA's Space Station Freedom (SSF) project that will serve as a platform to conduct long term scientific experiments in space. The LSM of SSF is aimed to carry out experiments related to space medicine, gravitational biology, genetics and biochemistry. FHL will also play an important role for many experiments being planned in manufacturing and biotechnology.

The FHL has been modelled and simulated by discrete event methodology in assigning responsibilities to an organized group of robots for routine handling of fluids. Its design is based on the discrete event sequence of units’ operations to be carried out to bring a real process from one initial state to a desired one. For example, the operation of a water sterilising unit used in FHL may be specified as a chain of three events of filling a bottle with water, placing it in a heating spiral and removing it when the required temperature has reached. Scheduling of transition from one state to another is based on time-to-next-event values obtained from trajectories of the dynamic model. Each unit operation is associated with a set of sensors for detecting its initializing and goal states.
2.6.3 Manufacturing System Simulation Tool

The last real-time application of discrete event modelling and simulation described in this section concerns with a comprehensive computer-aided manufacturing system simulation tool [Gar86]. This tool has been reported by the author as one of the largest applications of discrete event modelling and simulation in industry. It is currently being used at Wright Peterson Air Force Base, USA, as a part of Integrated Computer Aided Manufacturing Program (ICAM). The ICAM performs simulation of complex manufacturing processes and provides information about work flow within the factory, raw material inventories, shipping of finished goods, etc. It includes a graphic language for representing systems, a database for maintaining system configuration and a simulator for analysing system performance.

2.7 Summary

The concept of DEDS and the modelling approaches used for it were introduced in this chapter. There are many modelling approaches available for DEDS, each with its own merits, limitations and applications. A modelling approach that best describes a DEDS may be inappropriate for another DEDS. So far no DEDS model is mathematically as concise or computationally as feasible as the differential equations used for CVDS.
3.1 Introduction

This chapter describes the discrete event modelling of Distributed Manipulation Environment (DME). DME is a relatively new and unique concept in its aims, approach and underlying principles. It introduces a systematic, modular and distributed manipulation system for flexible automation and manufacturing. The discrete event modelling of DME is based on Discrete Event System Specifications (DEVS) formalism and is intended to capture the dynamics and concurrent behaviour of the DME system. The DME model developed in this chapter will serve as the basis for developing simulation models of DME in chapter 4 and an event-based controller in chapter 5.

3.2 Discrete Event Modelling of DME

A DME system is inherently discrete and can be viewed as a dynamic system with discrete state space and piecewise constant state trajectories. The time instants at which input and state transitions occur are usually irregular. This is similar to the general definition given by Ramadge of a discrete event dynamic system [Ram89].
The operation of a DME system is specified as a chain of concurrent and sequential events. These events in DME are the state transitions of the manipulation modules and the entities being manipulated.

The discrete event modelling of DME is expressed by DEVS formalism that focuses on the instant changes of a set of variables as the result of event happenings [Con88], [Zie84]. This generates time segments that are piecewise constant but usually spaced unequally as the time intervals between event occurrences are not constant. The modelling of DME takes place at two different levels. They are called atomic and coupled models and are defined in the following sections.

### 3.2.1 Atomic Model

The building block of DME is the manipulation module. The atomic model (AM) in DME, therefore, specifies the manipulation behaviour of a single manipulation module and can be defined as follows:

\[ AM = < X, S, Y, \delta_{\text{int}}, \delta_{\text{ext}}, \lambda, \tau > \]

where

- \( X = \{ \text{Sensory} \text{ inp}, \text{Message} \text{ inp} \} \); the set of inputs
- \( S = \{ s_1, s_2, ..., s_n \} \), the sequential state set of a manipulation module. The parameters that determine the DEVS state of a system depend on the specific task being carried out and hence may vary from one application to another. Typically, a pair \((q, x)\) that relates the current state \((q)\) of a manipulation module to the current input value \((x)\) would represent state \(s_i\) in the sequential state set \(S\).
- \( Y = \{ \text{Control-Signal out}, \text{Message out} \} \); the set of outputs
- \( \delta_{\text{int}} = S \rightarrow S; \) the internal transition function. It specifies the next state to which the system will transit after the lapse of the time, defined by the time advance function, provided no external event occurs in the meantime.
- \( \delta_{\text{ext}} = Q \times X \rightarrow S; \) the external transition function that describes the system behaviour under the action of an input and
\[ Q = \{ (s, e) \mid s \in S, 0 \leq e \leq t_a(s) \}; \text{ the total set that relates a sequential state (} s_j \text{) and the time elapsed (} e_j \text{) in that state} \]

- \( t_a = S \rightarrow R^+_0, \infty \); the time advance function. It defines the time for which the system remains in a given state before it undergoes the next internal transition provided that there is no change in the inputs in the meantime.
- \( \lambda = S \rightarrow Y \); the output function that is used to generate an external output.

A manipulation module may be in active or passive phase at any instant of time. In the passive phase the time advance function, \( t_a \), is infinity and the manipulation module is locked to a physical location \( s_j \). The manipulation module will stay in such a state indefinitely until an external event influences it. The internal transition function needs not to be defined for this state.

Under the influence of an external event, the manipulation module switches to an active state that is governed by the internal transition function and the time advance function. Once active, the manipulation module undergoes internal state transitions. All these internal state transitions are spontaneous in nature, i.e., the time advance function, \( t_a \), is zero, and the manipulation module is then said to be in a transitory state. The internal transition function and the time advance function are both defined by the dynamics of the manipulation modules.

An external transition is defined based on external events linked to each manipulation module. Depending on the type of manipulation module and the sensory linked attached to it; the external events may be generated by a sensor or triggered by another manipulation module as a request to carry out a task. An external event generated by a sensor linked to a manipulation module can interrupt the transition of manipulation module and force it to a passive state of \( s_{int} \). The next state taken up by the system when influenced by \( \delta_{ext} \) depends on the present state, external input and the time that has elapsed in the current state.
The functions required in this model can be defined as rules or algorithms that can be easily simulated. Figure 3.1 shows an atomic model in state s for an elapsed time e. The remaining time \( \tau \), after which the next internal transition would take place, can then be easily calculated as \( t_a(s) - e \).

![Figure 3.1: An Atomic Model](image)

### 3.2.2 Coupled Model

A coupled model defines the manipulation behaviour of an ensemble of manipulation modules that are collaborating to carry out a certain task. In practice a coupled model (CM) describes how the atomic models of several manipulation modules can be combined to form a new model. It contains the following information:

- The set of manipulation modules operating in the coupled model
- The influencees of each manipulation module
- The set of input ports through which external events are received
- The set of output ports through which external events are sent
- The coupling specification defining the external input coupling and the external output coupling. In this part, the manipulation modules whose input or output ports are connected to the input/output ports of the coupled model are identified.

A coupled model is formally described as

\[
CM = \langle D, C, Select \rangle
\]

where
Chapter3  Discrete Event Modelling of DME

- \( D = \{ \text{MM}_A, \text{MM}_B, \text{MM}_C, \ldots, \text{MM}_N \} \); is the set of component manipulation modules
- \( C = \{ C_1, C_2, C_3, \ldots, C_n \} \); is a set of ensembles that describes an individual manipulation module including its working relationships with other manipulation modules in the coupled model

For each \( \alpha \) in \( C \)
- \( C_\alpha = \{ M_\alpha, I_\alpha, Z_\alpha, \beta \} \) where
  - \( M_\alpha \) is the atomic model of the manipulation module \( \alpha \)
  - \( I_\alpha \) is the set of influencees of \( \alpha \) and

for each \( \beta \) in \( I_\alpha \),
- \( Z_\alpha, \beta \) is the interface map of \( \alpha \) with its influencee manipulation modules in the system
- The Select has rules or algorithms used to determine which manipulation module is allowed to carry out the next event.

A coupled model may contain any number of manipulation modules. The manipulation modules present in the coupled model are independent and operate synchronously or asynchronously. At many occasions during system operation, these manipulation modules would also be required to work concurrently. During the concurrent operation of manipulation modules, the DME system moves to a state that is the same resultant state reached if all the manipulation modules would have operated serially.

Similar to the atomic model, the coupled model is built for simulation through writing various algorithms. A coupled model consisting of two manipulation modules, \( M_\alpha \) and \( M_\beta \), is shown in Figure 3.2.
3.3 DME System Model

A DME system model is modular and hierarchical in nature and may be constructed recursively from its component models, each being atomic or itself a coupling of atomic models. The atomic models are coupled to produce a coupled model according to a coupling specification that varies for different applications. The coupled model thus obtained may in turn be used as a component model to be coupled with other component models in a larger multi-component system to give rise to a modular and hierarchical model construction. It is, therefore, not necessary that a coupled model should contain atomic models only as its constituents. Hence as the system size grows and the number of interacting sub-systems increases, more and more coupling layers are added to the overall system model as shown in figure 3.3.

In a larger multi-component system, the coupled model when used as a component model, would have the following structure.

- **INPUT** = The cross product of input sets of those manipulation modules which have their influencers outside the coupled model.
A1 Overall System Model
A21, A22, A231, A2321, A2322, A233 Atomic Models
A23 A Coupled Model of A231, A232 and A233
A232 A Coupled Model of A2321 and A2322
Cn stands for Coupling Specifications

Figure 3.3: A Modular DME System
- **OUTPUT** = The cross product of output sets of those manipulation modules that act as influencers outside the coupled model
- **STATES** = The cross product of manipulation modules' state set
- **TRANSITION FUNCTION** = The resultant of the transition functions of all individual manipulation modules in the coupled model, i.e.,
  \[ \delta_{\text{resultant}} = \delta_n(q_n, ..., \delta_2(q_2, \delta_1(q_1, x_1)), ...) \]
- **OUTPUT FUNCTION** = The resultant of the output functions of all individual manipulation modules in the coupled model

The state of the overall DME system, \( Q_{\text{system}} \), at any instant of time is a vector of total states of all its components. The system moves into a new state whenever one or more manipulation modules in a component model move to a new state. This system state can be specified as

\[ Q_{\text{system}} = (....(s_m, e_m)....) \]

The above expression suggests that at any given instant \( t \), each component, \( m \) in the system has been in state \( s_m \) for the lapsed time of \( e_m \). Since the time advance in state \( s_m \) is given by \( t_{am}(s_m) \), the component \( m \) is scheduled for an internal transition at time \( t + (t_{am}(s_m) - e_m) \). The next system state transition will now occur at a time which is the minimum of these scheduled times. Thus if the minimum of the residual time \( (t_{am}(s_m) - e_m) \) over the components \( m \) is \( \tau \), the next transition will occur at time \( t + \tau \).

The overall discrete model of a DME system is, therefore, a coupling of a set of component models. The component models assist to create a distributed hierarchically structured model of the system identical to the actual system. They also simplify the task of development, debugging and maintenance of the whole model.

### 3.4 DME Model Example

The discrete modelling procedure mentioned for DME is applied to the system shown in Figure 3.4 to illustrate the application and potential of this technique. In this system a
drilling task conventionally requiring a four-degrees of freedom robot is represented by a simple DME.

Here X is a plate on which a number of holes are to be drilled at different locations. The manipulation modules MM_A and MM_B are linear actuators used to control the plate X. A composite manipulation module attached to a drill bit is represented by CM1. This coupled model consists of a linear manipulation module MM_C and a rotary manipulation module MM_D and is capable of drilling holes in the plate at required positions. It is assumed that MM_A and MM_B move in steps rather than continuously and that all the points where holes are to be drilled are achievable. Thus to have a fine control over the whole work-space, MM_A and MM_B should have adequately large number of small steps. The accuracy of these manipulation modules is also important as the precision of the whole system depends on it.
Once the drilling head is located at the desired position through manipulation of MM_A and MM_B, CM1 is activated to drill a hole. The block diagram illustrating the interaction of the manipulation modules in this system is shown in Figure 3.5.

![Manipulation Modules in Drilling Model](image)

**3.4.1 Model of Drilling System**

The model of the drilling system is a coupled model based on MM_A, MM_B and CM1 as defined below:

\[
\text{Drill} = < \text{D, C, Select} >
\]

\[
\text{D} = \{ \text{MM}_A, \text{MM}_B, \text{CM1} \}
\]

\[
\text{C} = \{ \text{C}_A, \text{C}_B, \text{C}_{\text{CM1}} \}
\]

\[
\text{C}_A = \{ \text{M}_A, \text{I}_A, \text{Z}_{A,\text{CM1}} \}
\]

\[
\text{M}_A = < X_A, S_A, Y_A, \delta_{\text{intA}}, \delta_{\text{extA}}, \lambda_A, t_{A} >
\]

\[
\text{I}_A = \{ \text{CM1} \}
\]
\( Z_{A,CM1} = S_A \rightarrow X_{CM1} \) where
\( X_{CM1} = \{ (a,b) \mid a \in X_C, b \in X_D \} \)

\( C_B = \{ M_B, I_B, Z_{B,C} \} \)
\( M_B = < X_B, S_B, Y_B, \delta_{intB}, \delta_{extB}, \lambda_B, t_B > \)
\( I_B = \{ CM1 \} \)
\( Z_{B,C} = S_B \rightarrow X_{CM1} \)

\( C_{CM1} = \{ M_{CM1}, I_{CM1}, Z_{CM1,A}, Z_{CM1,B} \} \)
\( M_{CM1} = < D1, C1, Select1 > \) where
\( D1 = \{ MM_C, MM_D \} \)
\( C1 = \{ C_C, C_D \} \)

\( C_C = \{ M_C, I_C, Z_{CD} \} \)
\( M_C = < X_C, S_C, Y_C, \delta_{intC}, \delta_{extC}, \lambda_C, t_C > \)
\( I_C = \{ MM_D \} \)
\( Z_{CD} = S_C \rightarrow X_D \)

\( C_D = \{ M_D, I_D \} \)
\( M_D = < X_D, S_D, Y_D, \delta_{intD}, \delta_{extD}, \lambda_D, t_D > \)
\( I_D = \{ \} \)

\( I_{CM1} = \{ MM_A, MM_B \} \)
\( Z_{CM1,A} = S_{CM1} \rightarrow X_A \)
\( Z_{CM1,B} = S_{CM1} \rightarrow X_B \) where
\( S_{CM1} = \{ (a,b) \mid a \in S_C, b \in S_D \} \)

The individual atomic models of this drill example are given as
\( M_A = < X_A, S_A, Y_A, \delta_{intA}, \delta_{extA}, \lambda_A, t_A > \)
\( X_A = \{ v_1, v_2, \ldots, v_8 \}; \) speed set of MM_A such that
\( v_i \in \{ 0, +\alpha, -\alpha \} \) where \( \alpha \) is a constant so that MM_A moves with a uniform speed.

\( S_A = \{ (q_a, v_i) \} \) where \( q_a = \{ a_1, a_2, \ldots, a_8 \} \) (figure 3.4)

\( Y_A = \{ a_1, a_2, \ldots, a_8 \} \) (reaching at the desired position)

\( \delta_{intA}(q_a, v_i) = (q_a', v_i) \) where \( q_a' \) is the next state of MM_A
\[ \delta_{extA}(q_a, v_i, e, v_j) = (q_a, v_j) \quad (i, j \leq 8) \]

\[ \lambda_A = \{ a_1, a_2, \ldots, a_8 \} \text{ (produces output)} \]

\[ ta_A(s_i) = \text{the time advance function} \]

Manipulation modules MM_B, MM_C and MM_D have similar atomic models with the difference

\[ X_D = \{ \text{rotation} \mid \text{rotation} \in \{ 0, \beta \} \} \text{where } \beta \text{ is a constant so that MM_D rotates with a uniform speed} \]

\[ S_B = \{ (q_b, v_i) \} \text{ where } q_b = \{ b_1, b_2, \ldots, b_7 \} \text{ (figure 3.4)} \]

\[ S_C = \{ (q_c, v_i) \} \text{ where } q_c = \{ \text{home_position}, \text{contact_with_plate} \} \text{ and} \]

\[ S_D = \{ (q_d, \text{rotation}) \} \text{ where } q_d = \{ \text{start}, \text{end} \} \]

The last two expressions show that contrary to MM_A and MM_B whose state sets depend on the number and location of holes to be drilled, both MM_C and MM_D have only two states as they always perform a fixed job.

The parameters used in this example correspond to the definition of the models given in section 3.3.

### 3.4.2 The Select Algorithm

The task data including work-space size, the number of holes to be drilled, the location of the holes on the plate, system constraints, and similar information is fed into the task planner. The task planner, based on the system configuration and job accomplishment scheme, generates event labels for each manipulation module.

The scheduling algorithm employed in the selection scheme aims to minimise the manipulation time for the whole system in order to increase the productivity and reduce the cost. Genetic Algorithm (GA) is used to carry out this optimization. On the basis of the information produced by GA, the task planner generates event labels for different manipulation modules in the system. A more detailed study of this topic is carried out in chapter 5.
3.5 Summary

The application of discrete event modelling to Distributed Manipulation Environment was demonstrated in this chapter. The model developed is quite generic and can be applied to any type or configuration of DME. It provides description of a system at both atomic and coupled levels.

The feasibility of the developed model was demonstrated by an example. A drilling station based on the concept of DME was set up. The atomic and coupled models of the station were developed. The discrete event description of the operation of the station in terms of the state sets of manipulation modules was made. The selection algorithm used for scheduling the events was to minimise the overall timing of the operation.
CHAPTER 4

DISCRETE EVENT SIMULATION OF DISTRIBUTED MANIPULATION ENVIRONMENT (DME)

4.1 Introduction

Discrete event simulation is a broad discipline which has been successfully applied in many areas especially where analytical models are unsolvable due to the complexities or intractable statistical calculations. The emphasis of this chapter is on the discrete event simulation of DME. Discrete event models for DME were developed in the previous chapter. Those models serve in this chapter as the basis for developing simulation models for DME. Initially elements of discrete event simulation for DME will be produced. Then the simulation methodology developed for DME will be studied. Finally, the simulation models will be described.

4.2 Elements of a DME Simulation Model

The simulation of a DEDS (Discrete Event Dynamic System) may have a number of objectives [Rig89], [Tan90]. These objectives have an impact on how the simulation model and its elements should be developed. For a DME environment which is discrete event by nature and is basically meant for flexible manufacturing and automation, the system simulation model consists of the following interrelated modules.
• Component Modules
• Control Modules
• Utility Module
• Experimental Module

4.2.1 Component Module

A component module represents an atomic or coupled model discussed in the previous chapter. It consists of a set of instructions incorporating the logical functions and producing its behaviour. Once activated, it evaluates the state and output of the system at appropriate instants of the simulated time and updates them accordingly.

Since a component module depicts the function of an atomic or coupled model, the number of these modules depends on the size and the structure of the system being simulated. These modules communicate and cooperate with each other according to the logic of the control module so as to form a network of integrated modules capable of representing the whole system. In DME simulation model, component modules are expressed in terms of atomic and coupled simulators.

4.2.2 Control Module

The control module performs functions concerned with the simulation timing and integration of the component modules in the overall simulation model [Fis78]. For the purpose of system timing, it determines the time after which the next event should happen and the component modules that have to update their states as a consequence of happening that event. A new system state is thus generated. To identify the next event, the control module calculates the time interval for each exogenous variable of each component module to change its value. The shortest of all such intervals is the interval to the next event. After the simulation time has been advanced to the next event time, the component modules affected by that event, update their states and afresh estimations are
made as to when the component modules have to change their states. Figure 4.1 shows this timing arrangement that is at the heart of the discrete event simulation model.

```
INITIALIZE:
  Time = 0
  All Component Modules = Idle

Yes
  Time >= T
  Stop

FOR EACH MODULE:
  Evaluate Status (Active/Passive)
  Evaluate Time_Status

NEXT TIME STEP:
  Time_step = Min (Time_Status)
  (of All Modules)

NEW TIME:
  Increase Time by Time_STEP

FOR EACH MODULE:
  Decrease Time_Status(Module)
  Evaluate Its State, Outputs

STOP
```

Figure 4.1: The Control Module
The coordination among different component modules is also carried out by the control module. It is accomplished by communicating the values of previously calculated component modules' output to any other component modules having these outputs as inputs. To perform its function a control module may consist of many routines.

### 4.2.3 Utility Module

The utility module contains a number of routines that perform general purpose application independent functions. Among these utility routines are the various computational algorithms including statistical computations and numerical methods required for various sub-functions of component modules (atomic and coupled simulators) especially those of high-level coupled simulators. Utility routines are also required for control module and experimental module.

### 4.2.4 Experimental Module

The experimental module defines the experimental frame for the simulation model at hand. An experimental frame, according to Ören and Ziegler, defines a simplified state of affairs and a limited set of circumstances under which a system is subjected to experimentation [Öre79]. An experimental frame thus includes an observation module and an environment module.

An experimental module, therefore, concentrates on the frame defining information which depends on the nature of the specific simulation considered.

### 4.3 Discrete Event Simulation Methodology for DME

The simulation of a DME system at functional level is quite different from that of electrical networks or mathematical models derived from system characteristic equations. The discrete event simulation methodology for DME focuses on events which move the system from one state to the next, and assumes that nothing of importance
takes place between the two consecutive events [Zie91a]. In this process the following points are of interest:

- The prior state of DME
- The time of occurrence of the current event
- Selected time advance function which is in fact the minimum of the time advance functions of all the events that may occur in the current state
- The set of constraints, if any
- The next state of DME

Let a DME system be initialized to state $s_0$ at time $t_0$. This initialization can be expressed as

prior state = unspecified

\[ t = t_0 \]

selected time advance = unspecified or $\phi$

constraint set = null

next state = $s_0$

As this system evolves in time under events, its behaviour at time $t$ and in state $s$ will be represented as

prior state = $s$

\[ t = t \]

selected time advance = $t_a$

constraint set = $c$

next state = $s_n$ = unspecified

The next state $s_n$ of the system and new system parameters will be obtained as follows.

a) On the basis of $s$ and $t_a$, perform simulation to get $s_n$, the next state.

b) Calculate all possible $t_a(s_n)$ for the internal events that may occur in state $s_n$.

c) Find $t_a(s_n)_{\text{min}}$, the minimum of all $t_a(s_n)$ calculated in (b).
d) Set the parameters as follows for the calculation of state $s_{n+1}$.

\[
\text{time} = t + t_a(s_n)_{\text{min}} \\
\text{selected time advance} = t_a(s_n)_{\text{min}} \\
\text{prior state} = s_n \\
\text{next state} = \text{to be calculated} \\
\text{constraint set} = c'
\]

Steps a, b, c and d are then repeated again from this state to find out the next state and the system parameters.

### 4.4 Discrete Event Simulation Model for DME

In previous chapter, Discrete Event Modelling of DME was carried out both at atomic and coupled level. The discrete event simulation models of DME are based on discrete event mathematical models. A simulator is now being defined for each atomic and coupled model that incorporates the simulation behaviour into the mathematical model using the same set of functions and parameters specified in the atomic and coupled models.

Both the atomic and coupled simulators have static as well as dynamic behaviour [Pol91], [Cel90]. The static behaviour is expressed by defining the models in a proper format and in appropriate modules. The format mainly depends on the software implementing the simulation. The dynamic behaviour of the simulator is governed by the transition functions as explained in the next two sections.

#### 4.4.1 Atomic Simulator

An atomic model that defines the manipulation behaviour of a single manipulation module was defined as:

\[
\text{AM} = \langle X, S, Y, \delta_{\text{int}}, \delta_{\text{ext}}, \lambda, t_a \rangle
\]
Since it is a model that can not be further decomposed, $S_{AM}$, the simulator assigned to this atomic model has the following structure and is intended to represent a segment of the domain under study.

\[
\begin{align*}
S_{AM} \\
\text{operation} \\
\text{data} \\
\text{interfaces}
\end{align*}
\]

*Operations* operate on and manipulate the *data* to incorporate dynamics into the simulator while *interfaces* are used to communicate with the environment.

The atomic simulator has two state variables and three storage cells as shown in Fig 4.2.

The state variables are $S$ and $t_L$. $S$ represents a sequential state while $t_L$ indicates the time of the last event.

The three storage cells are denoted by $t_N$, $e$ and $\tau$ and respectively store information regarding the time of the next event, elapsed time since the last state transition and the time remaining for the next state transition to occur. Taking $t$ as the global time, the contents of the storage cells are well defined as

\[t_N = t_L + t_{d(s)}\]
The simulator interacts with its environment that includes other simulators in a predefined manner via an input and an output port. Through the input port it receives input and undergoes a state transition using either external or internal transition functions. The dynamic behaviour of the simulator is expressed as follows:

a) Case of $\delta_{\text{ext}}$ when receives an input $(x,t)$

where $x \in X$ and

t is global time

If $t$ satisfies $t_L \leq t \leq t_N$

$$e = t - t_L$$

$$s_n = \delta_{\text{ext}}(s,e,x)$$

$$t_L = t$$

$$t_N = t_L + \tau_a(s)$$

b) Case of $\delta_{\text{int}}$ when receives synchronisation signal to update state

If $t = t_N$

$$s_n = \delta_{\text{int}}(s)$$

$$t_L = t$$

$$t_N = t_L + \tau_a(s)$$

The simulator uses its output port to report $t_N$ to the control module that manages the simulation time of the whole system.
4.4.2 Coupled Simulator

A coupled model was defined previously as a coupling of a number of component models that work together to accomplish a certain sub-task and behave as a single sub-model in the whole system. A coupled simulator is a simulator associated with a coupled model and has the same structure as that of the coupled model. It consists of a number of component simulators, each responsible for a component model in the coupled model. These component simulators may be the atomic simulators or again coupled simulators of some other component simulators.

Mathematically a coupled simulator is defined as

\[ CM = < D, C, \text{select} > \]

where

- \( D \) represents the set of component models in the coupled model
- \( C \) represents the behaviour of a component model in the coupled model
- Select represents rules for the orderly behaviour of the model

Correspondingly the coupled simulator, \( S_{CM} \), will have the following structure.

\[ S_{CM} \]

- component simulators; atomic or coupled
- coupling scheme
- interfaces

A coupled simulator thus incorporates a coupling scheme in its structure to cater for the interfaces needed among its components. This coupling scheme is implemented by a coordinator which is responsible for coordinating and synchronising the component simulators within the coupled simulator and handling external events [Zie91]. Fig 4.3 shows a coupled simulator for \( n \) component simulators.
For each component model $D_i$ in the coupled model, there is a corresponding component simulator $S_i$ in the coupled simulator. Upon the receipt of an external event, the coordinator applies this input to all the component simulators attached to it which then behave as explained in section 4.4.1.

The dynamics of the coupled simulator is expressed as

a) Case of $\delta_{\text{ext}}$ when coupled simulator receives an input $(x,t)$

\[
\text{if } t_L < t < t_N
\]
send input \((x, t)\) to each component simulators

wait until all component simulators are done

\(t_L = t\)

\(t_N = t_N(\text{min})\) where \(t_N(\text{min})\) is the minimum of all \(t_N\) in \(S_{CM}\)

b) Case of \(\delta_{\text{int}}\)

At \(t = t_N\)

Select component simulator \(S_j\) with \(t_N(\text{min})\)

Apply \(\delta_{\text{int}}\) to \(S_j\)

send input \((x_{j,k}, t)\) to each influencee \(S_k\) of \(S_j\) where \((x_{j,k}, t)\) is the input to \(S_k\) from \(S_j\) obtained by using output function.

wait until simulator \(j\) and all of its influencees are done

\(t_L = t\)

\(t_N = t_N(\text{min})\)

For coupled simulator, a situation may arise when two or more component simulators want to utilize the same single resource at the same time while the resource cannot be shared. This situation can be handled by some tie breaking mechanism, the details of which will depend on the implementing software.

The external interface of the coupled model consists of an input and an output port. It receives input and synchronization signals through the input port and informs the control module about \(t_N(\text{min})\) through the output port. This interface structure is the same as that of the atomic simulator. Because of this, simulators may be combined in a modular and hierarchical fashion irrespective of the fact that a simulator at hand is an atomic or coupled in nature.
4.5 Simulation Model of Drilling System

A drilling hole system was set up in the previous chapter and its discrete event model was developed. The same system is considered here again for the purpose of discrete event simulation in DME, based on the criteria and methodology discussed in this chapter.

The discrete event drill model was composed of 4 atomic models (MM_A, MM_B, MM_C and MM_D) and one coupled model CM_i consisting of MM_C and MM_D. The atomic simulators, corresponding to these atomic models, have been expressed as processes. Each process contains a code specific to the nature of the manipulation module represented by the atomic model in the simulation model. The coupled model CM_i was expressed by a coupled simulator that was also implemented as a process, cm1.sim. It contains code regarding how MM_C and MM_D are coupled and behave when called by the system.

The process objects of the drill model enter into the simulation at an explicit time by the occurrence of some specific events. They become active either immediately or at a prescribed activation time. Each time a process is activated, it executes statements representing changes to the system state and then is terminated. Figure 4.4 shows the evolution of the simulation model through these processes [Rus90].

The information regarding how many and where holes are to be drilled on the plate is contained in two files h.dat and v.dat. These files act as a source of external events. For this example, data for 6 holes on the plate is contained in h.dat and v.dat. By changing data in these files, any number of holes can be drilled. The location of the holes can also be controlled. The speed of the manipulation modules and system simulation can also be set at the beginning of the task. Before starting drilling, marking of the points on the plates where holes are to be drilled according to the data contained in h.dat and v.dat is made. When a hole has been drilled at a marked position, it is also indicated.
The whole program for this drilling simulation model is contained in appendix A. The system state at various instants of times during the drilling process is illustrated through figure 4.5 to figure 4.11.
DME DRILL HOLE SYSTEM

DRILL MODEL

TIMESCALE = 100

LINEAR SPEED OF MANIPULATION MODULES (ft/min) = 1

ROTARY SPEED OF MANIPULATION MODULES (deg/sec) = 10

OK CANCEL

Figure 4.5: Setting up of Parameters

Figure 4.6: Initial Set Up

Figure 4.7: Marking Done
Figure 4.8: Mani. Modules Just Before 1st Hole

Figure 4.9: First Hole Drilled

Figure 4.10: Three Holes Drilled

Figure 4.11: Job Completed
4.6 Summary

A simulation methodology for developing simulation models for DME was discussed in this chapter. The developed simulation models use the same functions as described in the discrete event modelling of DME for scheduling interval events and executing transitions as a result of these internal events as well as external events.

A working simulation model for a drilling system was then developed using simscript and simgraphic. The simulation model has a routine for each manipulation module in the system besides some other routines for control and other general purposes. The model is quite flexible and can deal with any topology of the drill scheme.

The simulation models for manipulation modules used in the drill model are in fact generic in nature and can be used in other applications.
CHAPTER 5

EVENT-BASED
CONTROL OF
DISTRIBUTED
MANIPULATION
ENVIRONMENT (DME)

5.1 Introduction

The main objective of this chapter is to develop an event-based controller for DME. The event-based control paradigm is based on DEVS formalism. This helps in testing the controller through computer simulation before implementing it in real time.

During the course of this chapter event-based methodology in general will be described and its application to DME systems will be explained. To set the scene, a brief section is initially devoted to conventional control strategies to emphasis their difference from event-based control paradigm.

In order to demonstrate the viability of the method, the event-based control model for DME drilling system is then developed and simulated. This model generates control signals required for driving the manipulation modules operating the system. Finally, an optimization process developed to reduce the overall manipulation time of a DME is discussed.
5.2 Control Strategies

In every control strategy for automated manufacturing and process control, the system to be controlled is outfitted with a number of appropriate sensors. These sensors provide feedback signals on how the overall control system is behaving. In the light of sensor responses, the controller takes necessary steps by issuing commands to appropriate sub-systems to ensure that the system's evolution in time follows the required trajectory with admissible deviations [Lew94]. Figure 5.1 shows the basic configuration of a control system.

![Figure 5.1: Basic Configuration of a Control System](image)

5.2.1 Conventional Control

In conventional computer controlled systems, the output of the process is sampled regularly by a data acquisition sub-system. The readings of the sensors at each instant are stored and tested against the respective sensor windows, a sub-interval of the sensor output range [Luk86], [Bil89]. This comparison indicates how faithfully the system follows the desired behaviour. On the basis of this result, the controller issues corrective control actions to adjust the concerned sub-systems.

This conventional approach demands the sensors to be precise and reliable so that the readings acquired by the data acquisition sub-system and compared with the window values are accurately known. Figure 5.2 illustrates the required sensor attitude in
conventional digital control. At sampling interval T, the sensor reading should be within the $S_{\text{max}} - S_{\text{min}}$ window.

![Diagram](image)

Figure 5.2: Sensor Window in Conventional Control

### 5.2.2 Event-Based Control

In event-based control, each state transition of the system is associated with a definite time window [Zie89]. This time window is determined by the discrete event model of the system and usually varies from one state to another. The sensors are now assumed to respond to the controller within the time windows to confirm that the expected state transitions have occurred. The system moves from one state to another as long as the controller continues to receive the sensors’ responses within the expected time windows.

The event-based control logic, therefore, does not demand sensor output precision. The sensors are only required to have a finite number of states and thus can have threshold-type characteristics. Since this control logic depends on the comparison of the clock time with the window time, the whole burden of accuracy is placed on clock rather than on sensors' readings. Figure 5.3 shows the time window concept for an event-based control.
5.3 Event-Based Control Methodology

In chapter 3 the discrete event modelling of DME which is inherently a discrete system was discussed. In order to develop a broader methodology for event-based control, the discrete event modelling will be reviewed and a more generalised approach which can be applied to continuous dynamical system will be presented. In continuous dynamical systems the state is a continuous variable. In this method the current state is partitioned into blocks by defining boundaries. The process is similar to the multi-rate sampling of a continuous system. The DEVS model is then defined based on the boundaries created for the state. The defined boundaries are further introduced into the model and a new representation of the discrete model known as Boundary-Based Discrete Event Model is developed. This model is ultimately used to produce the event-based control model.

The overall methodology to develop an event-based control model for a dynamical system is thus a three-step process [Luh93]:

a) Work out the DEVS based discrete event model of the continuous state dynamical system to be controlled.
b) From discrete event model of the system, develop a boundary-based discrete event model.

c) Use boundary-based discrete event model to obtain event-based control model of the system.

5.3.1 Discrete Event Model of the Continuous Dynamical System

To obtain the discrete event model of the continuous dynamical system, the first assumption made is that the inputs to the system are piecewise constant time functions (e.g., sequences of a step function). Next, the continuous system is outfitted with a finite set of finite-states threshold-type sensors. These threshold-type sensors provide information about system state. For each state some of the sensors would be above the threshold while some others below. Thus the sensors divide the state space into a finite mutually exclusive state partitioning blocks. This is illustrated in figure 5.4 in which a single 4-state threshold-type sensor divides the state space into 4 partitioning blocks.

Figure 5.4: Partitioning of State Space by a 4-State Threshold-Type Sensor
The state of the system at a block is now represented by a pair \((q, x)\) where \(q\) is the current state of the dynamical system in that partitioning block and \(x\) is the current input.

The discrete event model of the system is then described as

\[
< X, S, Y, \delta_{\text{int}}, \delta_{\text{ext}}, \lambda, t_a >
\]

where

- \(X\) is the set of inputs
- \(S\) is the set of sequential states. A sequential state in this DEVS model is represented by the pair \((q, x)\) such that
  - \(q\) is the system state at the beginning of the state partitioning block's boundary and
  - \(x\) is the input at the boundary of the block
- \(Y\) is the set of outputs
- \(\delta_{\text{int}}\) is the internal transition function given as
  \[
  \delta_{\text{int}}(q, x) = (q', x) \quad \text{where} \quad q' \text{ is the system state at the next partitioning block}
  \]
- \(\delta_{\text{ext}}\) is the external transition function given as
  \[
  \delta_{\text{ext}}((q, x), e, x') = (q'', x') \quad \text{where} \quad q'' \text{ is the state captured by the dynamical system after receiving a constant input } x' \text{ for an elapsed time } e \text{ given by } 0 \leq e \leq t_a(q, x). \text{ The system, however, remains in the same partitioning block. As a result no immediate output is produced by } \delta_{\text{ext}}.
  \]
- \(t_a(q, x)\) is the time advance function. It indicates the time required to cross the current partitioning block containing \(q\), under an input \(x\)
- \(\lambda(q, x)\) is the output function that generates the output of the system as it enters the state \(q'\) at the next partitioning block after the time given by \(ta(q, x)\)

The above discrete event model is thus related to the original dynamical system with a homomorphism that faithfully preserves a correspondence between the states of the discrete event model and the original system under the corresponding transitions and output operations. The aim is not to capture all the internal structure of the system but the input-output behaviour with a greater degree of accuracy.
5.3.2 Boundary-Based Discrete Event Model

The above obtained discrete event model is then used to develop the boundary-based discrete event model. The boundary-based discrete event model is a special case of the general discrete event model [Kim93]. It is obtained from the discrete event model of the system by imposing the restriction that the inputs to the system can change only at the boundary crossing of state partitioning blocks. The state transitions are thus forced to occur only at boundary crossing of the partitioning blocks. The state notion (q, x) used in the discrete event model is then modified as (b, q, x) which represents the current partitioning block boundary b on which the state q resides with input x. The boundary-based model is defined as

$$< X, S, Y, S^{xt}, \delta_{int}, \delta_{ext}, \lambda, ta >$$

where

- X is the set of inputs; same as was for the discrete event model of the system
- Y is the set of outputs; as before
- $S \subseteq B \times \mathbb{R}^n \times X$ where B is a finite set of elements, each called boundary, of a real vector space $\mathbb{R}^n$ and $(b, q, x) \in S \Rightarrow q \in b$
- $\delta_{int}(b, q, x) = (b', q', x)$. The internal transition function of a boundary-based model thus takes the system state q on boundary b to system state $q'$ on another boundary $b'$ under the same input value x.
- $\delta_{ext}((b, q, x), 0, x') = (b, q, x')$. The external transition function now accommodates the restriction that the inputs to a boundary-based model can change only at the boundary crossings by making elapsed time $\epsilon = 0$. For the same reason, a change in the input of the boundary-based model does not cause an immediate change in the discrete state vector of the system.
- $\lambda(b, q, x) = \lambda(b, q', x) \forall q, q' \in b$. The output function for the boundary-based model shows that all elements of a boundary produce the same output. It is due to
this reason that the sensors used are of threshold-type whose outputs change only at boundary crossing.

• $ta(b, q, x)$ is the time advance function; the same as defined before.

A boundary-based discrete event model as described above thus represents a model in which events, either external or internal, occur at qualitatively significant points along the continuous state trajectory.

### 5.3.3 Event-Based Control Model

A boundary to boundary transition in a boundary-based discrete event model was defined as $\delta_{int}(b, q, x) = (b', q', x)$. It shows that all system state $q$ in a boundary $b$ goes to the next boundary $b'$ under the same input $x$. The inputs in a boundary-based model, as explained earlier, are not allowed to change within the boundary crossings. The boundary to boundary transition/behaviour thus becomes independent of $q$ and is determined by $b$ and $x$ only. This fact is utilized in the event-based control model of the system along with the concept of time window [Wan90]. The event-based control model is abstracted from boundary-based model of system and is expressed as [Luh93]

$$< X, S, Y, \delta_{int}, \delta_{ext}, \lambda, ta >$$

where

• $X$ is the set of input as before

• $S = B \times X$. The state pair $(b, x) \in S$ represents a section of contiguous states defined by the boundary $b$ that the system has reached and the input $x$ that is expected to take the system state on $b$ to the next boundary and is termed as phase.

• $Y$ is the finite set of output

• $\delta_{int}(b, x) = (b', x)$. The internal transition function, as before, has no effect on the stored input and is accompanied by a change in boundary.

• $\delta_{ext}((b, x), 0, x') = (b, x')$. The external events thus have no immediate effect on the boundary in which the system is currently residing.

• $ta: S \rightarrow R^+_{0, \infty} \times R^+_{0, \infty}$ so that $ta(s) = (t_1, t_2)$ with constraints that
t_2 \geq t_1 \text{ and } t_2, t_1 \in \mathbb{R}_0^+. \text{ The time advance function in the event-based control model does not define a definite time as was the case with the previous models. It now introduces a time window within which the controller expects the sensors to respond to confirm the crossing of the expected boundary. The time window normally differs from boundary to boundary and is based on the time and its normal deviation required to reach the next boundary.}

- \lambda(b, x) \text{ is the output function as defined before}

Figure 5.5 shows this three-stage process of developing an event-based control model of a continuous dynamical system [Luh93].

5.4 Event-Based Control Model Approach for DME

Before developing an event-based control model for DME using the procedure described in the previous section, the specific nature of DME needs to be considered. A DME system has a distributed architecture. This distributed nature is also reflected in its control scheme. Hence an event-based control model will be developed for each subsystem. These event-based control models will run concurrently and communicate and coordinate with each other so that the overall system evolves in a smooth and disciplined manner.

Since basic units of a DME system are the manipulation modules, the control of the system is achieved by controlling the manipulation modules. Each manipulation module in the system is, therefore, outfitted with a threshold-type finite-state sensor though finite set of bi-level sensors may also be used. All such sensors divide the state space into partitioned blocks forming a cellular grid structure as shown in figure 5.6.
Chapter 5

Event-Based Control of DME

71

Output Partition Blocks

Continuous State Model

Boundary-Based Discrete Event Model

Event-based Control Model

Legend

\(x_1, x_2\) Inputs
\(b_1, b_2, b_3\) State Partitioning Boundaries
\(q_1, q_2, q_3\) Current System States of Dynamical System
\((b, x)\) State of Event-Based Control Model
\((b, q, x)\) State of Boundary-Based Model

\[\begin{array}{l}
\text{Internal Transition} \\
\text{External Input Event}
\end{array}\]

Figure 5.5: Event-Based Control Model
The size of the portioning block depends mainly on the specific job being carried out and indicates the range to be tolerated in the measurement of sensors. The threshold-type sensors are thus chosen according to the required width of the partitioning blocks. The crossing of the partitioning blocks in DME is considered as internal events and is modelled by internal transition function. The internal transition function along with the time advance function thus determines the time and state of the next boundary crossing. The changes in the input, e.g., speed control of manipulation module, are modelled as external events and are implemented by external transition function.

The manipulation modules in a DME system work independently and cooperate with each other to achieve set goals. After transition to a new state, a manipulation module normally waits for a definite time so that the other manipulation modules in the system may respond accordingly. This waiting time may be zero or a function of the time advance functions of other manipulation modules in the system.

To model this waiting time, the notion of wait state is introduced. Each state of a manipulation module is followed by the wait state that is used to model its waiting time. Thus for the purpose of event-based control, the states of a manipulation module can be represented as shown in figure 5.7.
The above figure shows that after crossing a boundary, a manipulation module waits for a time of $ta(w_n)$ before it is activated again to reach the next boundary. The total time taken by the manipulation module from one boundary to the next is, therefore, the sum of $ta(w_{n-1})$ and $ta(s_n)$ where $ta(s_n)$ is the time for which the manipulation module remains active in state $s_n$ under a given input.

The resulting control models of the manipulation modules are, therefore, interrelated with each other through the wait states and any malfunction of the system equally affects all these control models.

The time windows for different states are determined by parameter variations of specific job being carried out, under normal operating conditions. These can also be calculated by a series of system simulation runs by varying the parameters within the range of their normal tolerance.

### 5.5 Event-Based Control Model of Drilling System

The approach developed for the event-based control of DME is applied to drilling system to work out its control model which is then simulated on a computer. The DME model of drill hole system is expressed as

$$\text{Drill} < D, C, \text{Select} >$$

and consists of 2 manipulation modules MM_A and MM_B and a coupled model CM1 which decomposes into manipulation modules MM_C and MM_D. The event-based control model for each manipulation module is developed taking into consideration the
other manipulation modules in the system. These control models of all manipulation modules then collectively constitute the event-based model of the system.

5.5.1 Event-Based Control Model for MM_A

To develop an event-based model for MM_A, a finite-state threshold-type sensor is attached to it that divides its state space into partitioning blocks. As the boundary of a new partitioning block is reached during the movement of MM_A, sensor sends a confirmative single to the controller. During normal operation the controller, therefore, always knows when the new partitioning block is reached by MM_A and it then issues appropriate control commands in order to either place MM_A in a wait state for a time given by ta(wait state) or to move it to the next desired boundary. Figure 5.8 shows the input events and the state partitioning of the work-space with respect to MM_A. This is used to develop boundary-based discrete event model of MM_A.

![Diagram](image)

Legend

- **b_n** State Partitioning Boundary
- **x_n** Speed of MM_A
- **s_n** State of MM_A
- **w_n** Wait State of MM_A

Figure 5.8: State Diagram of MM_A for Event-Based Control
5.5.1.1 Discrete Event Model of MM_A

\[ \text{MM}_A = < X_A, S_A, Y_A, \delta_{\text{int}_A}, \delta_{\text{ext}_A}, \lambda_A, t_A > \]

where

- \( X_A \) is a set of external inputs, e.g., speed control of MM_A
- \( Y_A \) is a set of outputs, e.g., reaching at the required boundary
- \( S_A = (s_a, x_a) \); where \( s_a \) is the state of MM_A at the beginning of the partitioning block and \( x_a \) is input at that boundary
- \( \delta_{\text{int}_A}(s_{a1}, x_a) = (s_{a2}, x_a) \); where \( s_{a2} \) is the state of MM_A at the next desired block's boundary. The boundary crossings are modelled as internal events.
- \( \delta_{\text{ext}_A}(s_{a1}, x_{a1}, e, x_{a2}) = (s_{a2}, x_{a2}) \); the change in input modelled as external event
- \( \lambda_A(s_a, x_a) \) = output function that generates output \( y_a \) when a new boundary is reached under input \( x_a \)
- \( t_A(s) \) = time advance function in state \( s \) when MM_A is active
- \( t_A(w_n) = t_{\text{cm1}}(s_n) + (t_{\text{b}}(s_n) - t_A(s_n)) \) if \( t_{\text{b}}(s_n) > t_A(s_n) \)
- \( = t_{\text{cm1}}(s_n) \) if \( t_A(s_n) > t_{\text{b}}(s_n) \)

5.5.1.2 Boundary-Based Model of MM_A

Boundary-Based MM_A = < X_A, S_A, Y_A, \delta_{\text{int}_A}, \delta_{\text{ext}_A}, \lambda_A, t_A >

where

- \( X_A \) is a set of external inputs; the same as before in discrete event model
- \( Y_A \) is a set of outputs; the same as before in discrete event model
- \( S_A = (b_a, s_a, x_a) \); where \( b_a \) is the boundary where the state of MM_A is \( s_a \) and the input is \( x_a \)
- \( \delta_{\text{int}_A}(b_{a1}, s_{a1}, x_a) = (b_{a2}, s_{a2}, x_a) \); where \( b_{a2} \) is the next boundary at which state of MM_A is \( s_{a2} \)
- \( \delta_{\text{ext}_A}(b_{a1}, s_{a1}, x_{a1}, 0, x_{a2}) = (b_{a1}, s_{a1}, x_{a2}) \)
- \( \lambda_A(b_{a1}, s_{a1}, x_{a1}) = \lambda_A(b_{a1}, s'_{a1}, x_{a1}) \) for all \( s_{a1}, s'_{a1} \in b_{a1} \)
• \( t_a(s) \) = time advance function as defined before
• \( t_a(w_n) \) = time advance function for wait state as defined before

### 5.5.1.3 Event-Based Control Model for MM_A

Finally, the event-based control model for MM_A is given as

\[
\text{Event-Based MM}_A = \langle X_A, S_A, Y_A, \delta_{\text{int}A}, \delta_{\text{ext}A}, \lambda_A, t_aA \rangle
\]

where

- \( X_A \) = A set of external inputs; the same as before
- \( Y_A \) = A set of outputs; the same as before
- \( S_A = B_A \times X_A \)
- \( \delta_{\text{int}A}(b_{a2}, x_a) = (b_{a2}, x_a) \); where \( b_{a2} \) is the next boundary crossing
- \( \delta_{\text{ext}A}(b_{a1}, x_{a1}), 0, x_{a2} = (b_{a1}, x_{a2}) \)
- \( A_{\text{ext}}(b_{a1}, x_{a1}) = \) output function
- \( t_a(c) = (t_2 - t_1) \) with \( t_2 > t_1 \) and \( t_1, t_2 \in \mathbb{R}^+_0,\infty \)
- \( t_a(w_n) = (w_{t2} - w_{t1}) \) where
  \[
  w_{t2} = \max(t_a(w_n)) \text{ and } w_{t1} = \min(t_a(w_n))
  \]

The event-based control models for MM_B, CM1, MM_C and MM_D are derived in the same way and are given in appendix F.

### 5.5.2 Simulation of Event-Based Controller of Drilling System

The event-based control models for drilling system are simulated using Simscript to generate control/actuation signals for MM_A, MM_B, CM1, MM_C and MM_D. These control signals activate the manipulation modules to move to the next partitioning block. It is assumed that during the normal operation:

- MM_A takes 1 unit of time to move the plate by a distance of \( \frac{\text{plate_height}}{(Y_{\text{max}} + 1)} \). The variable \( Y_{\text{max}} \) is the maximum of the y-coordinates of the locations of all the holes to be drilled.
• MM_B takes 1 unit of time to move the plate by a distance of (plate_width/(X_{max}+1)). The variable $X_{max}$ is the maximum of the x-coordinates of the locations of all the holes to be drilled.

• MM_C takes 1.5 unit of time in locate the drill bit on the plate while MM_D takes 1.5 unit of time to drill a hole.

The time windows in this example are assumed to have a width of ±10\% of the ideal time required to cross the partitioning blocks. In case a sensor response is not received within its time window, the whole system operation is halted and a diagnostic message telling which manipulation module has caused the problem is issued. This is shown by following three simulation results in which control signals for the first three holes are produced.

A successful operation of the system is illustrated in figure 5.9 in which the controller receives all the sensor responses within time windows and hence issues actuation signals to manipulation modules to complete the task.

![Figure 5.9: Control Signals - Normal System Operation](image-url)
Figure 5.10 shows the case when MM_A does not work properly and the sensor response is received by the controller before the time window. The controller then issues the error message and stops generating of actuation signals to manipulation modules for the rest of the job. In figure 5.11 another situation is indicated in which CM1 malfunctions and the sensor response is received by the controller after the time window. Finally, figure 5.12 shows the relationship of the control signals of MM_C and MM_D with the control signal of their coupled model, CM1. Complete code for this control model is contained in appendix B.

Figure 5.10: Control Signals - MM_A Faulty
Figure 5.11: Control Signals - CM1 Faulty
5.6 Job Optimization in DME

In event-based control strategy of DME, the controller moves the system through a predefined succession of state partitioning boundaries to accomplish the desired task. The sequence in which these state partitioning boundaries are traversed depends on how the job has been scheduled. In most manufacturing and production applications, information regarding available resources, system configuration, system constraints, end objectives, etc., are clearly known. Besides, a DME system, similar to other DEDS usually performs some repeatable task.

The task to be performed by a DME system can be scheduled optimally to minimise the overall manipulation time and hence to increase productivity and reduce cost. The concept of task planner based on Genetic Algorithm (GA) is introduced to achieve this.

Figure 5.12: Relationship Between Control Signals of MM_C, MM_D and CM1
objective. The task planner takes the necessary data and based on the system configuration, generates the optimized sequence of operations for the given task. Before discussing the algorithm employed by the task planner for the job scheduling, it would be useful to describe briefly the characteristics and operation of genetic algorithm.

5.6.1 Genetic Operators

The three basic genetic operators used in genetic algorithm are:

- Crossover
- Mutation
- Inversion

5.6.1.1 Crossover

In GA, reproduction takes place by crossover operation. During crossover, two selected parents swap a portion of their structures to generate two new off-springs. The portion of the structure to be swapped depends on the cross over point or points that are determined by random selection. Figure 5.13 shows the usual single-point crossover operation for parents having a binary representation.

![Figure 5.13: The Crossover Operation](image)

The crossover operation thus provides new points for further testing in the search of global maxima and is normally assigned a high probability.
5.6.1.2 Mutation

The mutation operator is used to change randomly selected elements of an off-spring according to a specified probability. It ensures that all the points in the search space remain reachable, and acts as a local search close to the current point. Figure 5.14 shows the mutation operation.

![Mutation Operation](image)

Figure 5.14: The Mutation Operation

5.6.1.3 Inversion

Inversion operator is applied to an off-spring for inverting the order of elements of a selected segment of its structure. The size and the location of this segment depend on two index points that are chosen at random. Inversion provides more fertile ground for searching a better point. Figure 5.15 illustrates the inversion process.

![Inversion Operator](image)

Figure 5.15: Inversion Operator

5.6.2 Application of GA to Problems

Following are the steps involved in solving a problem by GA.
**Chapter 5  
Event-Based Control of DME**

**i) Encoding**

The first step is to determine an appropriate encoding for the problem as GA works with the encoded parameters rather than the actual one. For this, the search space of the problem is mapped on to a set of finite strings over finite alphabet such that each point in the search space is represented by exactly one such string called chromosome. The length of chromosomes is generally equal to the product of the number of variables involved in the optimization process and the length of every variable encoded in some suitable format [Dav91].

**ii) Initial population**

An initial population of chromosomes is then selected, usually at random. GA then starts its parallel search for global maxima from this population. The size of the population is a free parameter and is a compromise between the convergence of the search space and the time required to produce the next generation.

**iii) Fitness**

A fitness is calculated for each chromosome in the population according to a suitable evaluation function. The fitness value indicates the relative nearness to optimum of an individual as compared to others. An individual with higher value of fitness has more probability to survive and contribute one or more off-springs in the next generation.

**iv) Parent selection**

The more fit individuals are selected to act as parents according to a noisy selection, i.e., a parent is chosen at random but with a probability that corresponds to its relative fitness.

**v) Crossover**

The selected parents are crossed according to a predefined crossover probability to produce their off-springs.
vi) **Mutation**

Mutation operator is then applied to off-springs with a probability that is normally very small. As a result small random changes occur in a few randomly chosen individuals.

vii) **Inversion**

Inversion operator is also applied to off-springs according to a specified probability. Consequently, a few randomly selected individuals undergo reordering within their own structure.

viii) **Next Generation**

The fitness of the off-springs is calculated. The off-springs must have fitness value greater than the parents to go into the next generation. Otherwise, parents are copied to the next generation. The process now iterates itself from step (iii) until the desired degree of convergence is achieved.

### 5.6.3 Job Scheduling Algorithm

Job scheduling in manufacturing environment is basically a combinational problem that is solved by the search of the best permutation of the sub-tasks to be performed. Genetic algorithm serves this purpose of finding the best permutation. The problem resembles to some extent to the travelling sales man problem already treated in the literature (e.g., [Gre87]).

The nature of the problem suggests a non-binary string (chromosome) style and therefore decimal coding has been used for this problem. Moreover, classical crossover operators, both single-point crossover and two-points crossover, may duplicate some of the sub-tasks and eliminate the others in the resulting off-springs thus constructing illegal sequence of sub-tasks. One method to deal with such a situation is the penalty approach whereby a penalty is added to an infeasible solution so that the probability of its survival and contributing to the next generation is greatly reduced [Fal91]. The penalty approach for infeasible solution is, however, not suitable for job optimization.
like problems and the crossover operator should therefore always construct legal solutions. The crossover operator used in the optimization problem only modifies the order of the sub-tasks in order to produce a valid solution and is based on the guidelines suggested by Lin [Lin93] and is described below:

**Step 1:**


**Step 2:**

Randomly draw two indexes, $p_1$ and $p_2$, to serve as the crossover points. Then, $\text{Child}_1[p_1 : p_2] = \text{Parent}_1[p_1 : p_2]$, and $\text{Child}_2[p_1 : p_2] = \text{Parent}_2[p_1 : p_2]$

**Step 3:**

Initialize two matching vectors and set their corresponding indexes. That is,

$\text{Mating}_1[1 : n] = \text{Mating}_2[1 : n] = 0$

$\text{Mating}_1[\text{Parent}_1[p_1 : p_2]] = \text{Parent}_2[p_1 : p_2]$ and

$\text{Mating}_2[\text{Parent}_2[p_1 : p_2]] = \text{Parent}_1[p_1 : p_2]$

**Step 4a:**

For each Child$_1[i] = 0$, $1 \leq i \leq n$, perform the following steps:

$k = \text{Parent}_2[i]$

while $(\text{Mating}_1[k] \neq 0); k = \text{Mating}_1[k]$

$\text{Child}_1[i] = k$

**Step 4b:**

For each Child$_2[i] = 0$, $1 \leq i \leq n$, perform the following steps:

$k = \text{Parent}_1[i]$

while $(\text{Mating}_2[k] \neq 0); k = \text{Mating}_2[k]$

$\text{Child}_2[i] = k$

The usual mutation operator is also not suitable for job scheduling in DME as a random change of a sub-task is bound to generate an illegal solution. The mutation operator is,
therefore, not used for job optimization. Instead, inversion operator is employed to serve the purpose of random perturbation in genetic algorithms.

To apply optimization to job scheduling in DME, the sub-tasks that have no precedence constraints are identified. For example in drilling system, overall optimization can be achieved by finding the most efficient way of handling work-space by MM_A and MM_B for various holes at different locations. CM1 always performs the same fixed sub-task and does need not to be included in the optimization process. After optimizing the process, task planner generates event labels for different manipulation modules in the system specified in term of order of the sub-tasks to be performed. The event lists produced by the task planner are sent to respective manipulation modules prior to the operation and the next state partitioning boundary is determined by event-based controller in accordance with these event lists. Figure 5.16 shows the optimized job scheduled determined by the task planner to drill 10 holes on the plate. The code for job optimization in DME has been developed using C++ and is provided in appendix E.

![Figure 5.16: Job Optimization](image)

The role of the task planner is not that of a supervisor, though it may keep record of all event happenings during system operation. This record along with the error message
generated by the event-based controller can greatly help in system diagnosis in case of any malfunction.

5.7 Summary

In this chapter, event-based control methodology was discussed and applied to DME systems. The feasibility of the developed event-based controller was studied by applying it to the drilling system. It was shown that the event-based controller of DME can be validated before implementation by computer simulation. A job optimization algorithm, aimed at minimising the overall system timing, has also been developed.
CHAPTER 6

CASE STUDY:
EXTENDED DRILLING SYSTEM (EDS)

6.1 Introduction

The mathematical, simulation and control models for DME were developed in the previous chapters. The application of these models was also demonstrated through a simple drilling system in the respective chapters.

In this chapter, the methodologies developed will be further validated through a more complex case study. In this study, the drilling system is expanded as a complete production line with inlet and outlet conveyers. This system will be referred to as Extended Drilling System (EDS). The modelling, simulation and control system will be developed for all aspects of the system including the movement of parts/objects from inlet storage to the working area, performing the required task and then transfer of the finished products to the outlet storage. The EDS represents a typical production line and hence the developed models will demonstrate the feasibility and practicality of the methodologies developed.

In the course of this chapter, structure assumed for EDS will be described. The modelling will be carried out and simulation will be presented. Finally, event-based controller for the system will be designed and simulated.
6.2 Elements of EDS

The EDS is a distributed and concurrent manufacturing system of eleven manipulation modules which are divided into four coupled and one atomic models, as shown in figure 6.1.

In EDS, a plate is transferred from the inlet stack to the working area via a conveyor belt. Two different sizes of holes are then drilled on the plate according to the information contained in two data files. The data files are generated by the job optimization algorithm and have information on the number and size of the holes and their locations on the plate. After accomplishing the drilling task, the finished plate is
transferred to the store while another plate is transferred into the work area at the same instant. The EDS keeps working until all the plates in the inlet stack have been drilled and reached to the storehouse.

The coupled and atomic models used in EDS are mentioned below.

**CM1**

Coupled model CM1 consists of two prismatic manipulation modules; MM1 and MM2. It picks a plate from the inlet stack and deposits it on the conveyor belt to be transferred to the working area.

**CM2**

This coupled model performs a dual job.

- It off loads a plate from the inlet conveyor and moves it to the work area. It also transfers the plate back to the outlet conveyor after drilling.
- It also manipulates the drill bit and the plate during the drilling process and adjusts the plate where the hole should be drilled according to the data in the input file.

The CM2 coupled model consists of two prismatic manipulation modules MM3 and MM4 and a sub-coupled model CM21. The MM3 manipulation module transfers a plate to and from the conveyor. The position of the plate relative to the drilling bit is also adjusted by MM3 in conjunction with MM4. The coupled model CM21 executes the drilling process and consists of a prismatic manipulation module MM5 and a revolute manipulation module MM6 driving the drill bit.

**CM3**

The nature of operation of CM3 is similar to CM1. It picks the processed plates from the conveyor belt and transfers them to the storehouse. It also consists of two prismatic manipulation modules of MM7 and MM8.

**CM4**

As explained earlier, two different types of holes are drilled in each plate. The coupled model CM4 changes the drill bit contained by the coupled model CM21. Once, holes of the first size have been drilled, CM4 activates and replaces the drill bit by the second
size. It consists of a prismatic manipulation module MM9 and a revolute manipulation module MM10.

*MMA*

MMA is an atomic model representing a revolute manipulation module. Its function is to drive the conveyor belt at the appropriate instants during EDS operation.

The configuration and interconnections of these modules in EDS are shown in figure 6.2.

Figure 6.2: Configuration of Modules in EDS
6.3 Model of EDS

The model of EDS is a coupled model based on CM1, CM2, CM3, CM4 and MMA, and is defined below.

$$EDS = < D, C, Select >$$

where

$D = \{ CM1, CM2, CM3, CM4, MMA \}$

$C = \{ C_{CM1}, C_{CM2}, C_{CM3}, C_{CM4}, C_{MMA} \}$

$C_{CM1} = \{ M_{CM1}, I_{CM1}, Z_{CM1,\beta} \}$ where $\beta = I_{CM1}$

$M_{CM1} = \{ D1, C1, Select1 \}$

$D1 = \{ MM1, MM2 \}$

$C1 = \{ C_{MM1}, C_{MM2} \}$

$C_{MM1} = \{ MM1, I_{MM1}, Z_{MM1,\beta} \}$ where $\beta = I_{MM1}$

$MM1 = \{ X_{MM1}, S_{MM1}, Y_{MM1}, \delta_{intMM1}, \delta_{extMM1}, \lambda_{MM1}, ta_{MM1} \}$

$I_{MM1} = \{ MM2 \}$

$Z_{MM1,MM2} = S_{MM1} \rightarrow X_{MM2}$

$C_{MM2} = \{ MM2, I_{MM2}, Z_{MM2,\beta} \}$ where $\beta = I_{MM2}$

$MM2 = \{ X_{MM2}, S_{MM2}, Y_{MM2}, \delta_{intMM2}, \delta_{extMM2}, \lambda_{MM2}, ta_{MM2} \}$

$I_{MM2} = \{ \}$

$I_{CM1} = \{ CM2, MMA \}$

$Z_{CM1,CM2} = S_{CM1} \rightarrow X_{CM2}$

$Z_{CM1,MM4} = S_{CM1} \rightarrow X_{MM4}$

$C_{CM2} = \{ M_{CM2}, I_{CM2}, Z_{CM2,\beta} \}$ where $\beta = I_{CM2}$

$M_{CM2} = \{ D2, C2, Select2 \}$

$D2 = \{ MM3, MM4, CM21 \}$

$C2 = \{ C_{MM3}, C_{MM4}, C_{CM21} \}$

$C_{MM3} = \{ MM3, I_{MM3}, Z_{MM3,\beta} \}$ where $\beta = I_{MM3}$

$MM3 = \{ X_{MM3}, S_{MM3}, Y_{MM3}, \delta_{intMM3}, \delta_{extMM3}, \lambda_{MM3}, ta_{MM3} \}$

$I_{MM3} = \{ CM21 \}$

$Z_{MM3,CM21} = S_{MM3} \rightarrow X_{CM21}$

$C_{MM4} = \{ MM4, I_{MM4}, Z_{MM4,\beta} \}$ where $\beta = I_{MM4}$

$MM4 = \{ X_{MM4}, S_{MM4}, Y_{MM4}, \delta_{intMM4}, \delta_{extMM4}, \lambda_{MM4}, ta_{MM4} \}$

$I_{MM4} = \{ CM21 \}$

$Z_{MM4,CM21} = S_{MM4} \rightarrow X_{CM21}$

$C_{CM21} = \{ M_{CM21}, I_{CM21}, Z_{CM21,\beta} \}$ where $\beta = I_{CM21}$

$M_{CM21} = \{ D21, C21, Select21 \}$

$D21 = \{ MM5, MM6 \}$

$C21 = \{ C_{MM5}, C_{MM6} \}$

$C_{MM5} = \{ MM5, I_{MM5}, Z_{MM5,\beta} \}$ where $\beta = I_{MM5}$

$MM5 = \{ X_{MM5}, S_{MM5}, Y_{MM5}, \delta_{intMM5}, \delta_{extMM5}, \lambda_{MM5}, ta_{MM5} \}$

$I_{MM5} = \{ MM6 \}$

$Z_{MM5,MM6} = S_{MM5} \rightarrow X_{MM6}$

$C_{MM6} = \{ MM6, I_{MM6}, Z_{MM6,\beta} \}$ where $\beta = I_{MM6}$


Chapter 6 Case Study: Extended Drilling System

\[ \text{MM6} = \{ X_{\text{MM6}}, S_{\text{MM6}}, Y_{\text{MM6}}, \delta_{i\text{MM6}}, \delta_{e\text{MM6}}, \lambda_{\text{MM6}}, \text{ta}_{\text{MM6}} \} \]

\[ \text{I}_{\text{MM6}} = \{ \text{MM5} \} \]

\[ Z_{\text{MM6}, \text{MM5}} = S_{\text{MM6}} \rightarrow X_{\text{MM5}} \]

\[ \text{I}_{\text{CM21}} = \{ \text{MM3, MM4} \} \]

\[ Z_{\text{CM21, MM3}} = S_{\text{CM21}} \rightarrow X_{\text{MM3}} \]

\[ Z_{\text{CM21, MM4}} = S_{\text{CM21}} \rightarrow X_{\text{MM4}} \]

\[ \text{I}_{\text{CM2}} = \{ \text{CM3, CM4, MMA} \} \]

\[ Z_{\text{CM2, CM3}} = S_{\text{CM2}} \rightarrow X_{\text{CM3}} \]

\[ Z_{\text{CM2, CM4}} = S_{\text{CM2}} \rightarrow X_{\text{CM4}} \]

\[ Z_{\text{CM2, MMA}} = S_{\text{CM2}} \rightarrow X_{\text{MMA}} \]

\[ \text{CM3} = \{ \text{MCM3, I}\text{CM3}, Z_{\text{CM3, } \beta} \} \text{ where } \{ \beta \mid \beta \in \text{I}_{\text{CM3}} \} \]

\[ \text{M}_{\text{CM3}} = \{ \text{D3, C3, Select3} \} \]

\[ \text{D3} = \{ \text{MM7, MM8} \} \]

\[ \text{C3} = \{ \text{CM7, MM8} \} \]

\[ \text{CM7} = \{ \text{MM7, I}\text{MM7}, Z_{\text{MM7, } \beta} \} \text{ where } \{ \beta \mid \beta \in \text{I}_{\text{MM7}} \} \]

\[ \text{MM7} = \{ X_{\text{MM7}}, S_{\text{MM7}}, Y_{\text{MM7}}, \delta_{i\text{MM7}}, \delta_{e\text{MM7}}, \lambda_{\text{MM7}}, \text{ta}_{\text{MM7}} \} \]

\[ \text{I}_{\text{MM7}} = \{ \text{MM8} \} \]

\[ Z_{\text{MM7, MM8}} = S_{\text{MM7}} \rightarrow X_{\text{MM8}} \]

\[ \text{CM8} = \{ \text{MM8, I}\text{MM8}, Z_{\text{MM8, } \beta} \} \text{ where } \{ \beta \mid \beta \in \text{I}_{\text{MM8}} \} \]

\[ \text{MM8} = \{ X_{\text{MM8}}, S_{\text{MM8}}, Y_{\text{MM8}}, \delta_{i\text{MM8}}, \delta_{e\text{MM8}}, \lambda_{\text{MM8}}, \text{ta}_{\text{MM8}} \} \]

\[ \text{I}_{\text{MM8}} = \{ \text{MM7} \} \]

\[ Z_{\text{MM8, MM7}} = S_{\text{MM8}} \rightarrow X_{\text{MM7}} \]

\[ \text{ICM3} = \{ \} \]

\[ \text{CM4} = \{ \text{MCM4, I}\text{CM4}, Z_{\text{CM4, } \beta} \} \text{ where } \{ \beta \mid \beta \in \text{I}_{\text{CM4}} \} \]

\[ \text{M}_{\text{CM4}} = \{ \text{D4, C4, Select4} \} \]

\[ \text{D4} = \{ \text{MM9, MM10} \} \]

\[ \text{C4} = \{ \text{CM9, CM10} \} \]

\[ \text{CM9} = \{ \text{MM9, I}\text{MM9}, Z_{\text{MM9, } \beta} \} \text{ where } \{ \beta \mid \beta \in \text{I}_{\text{MM9}} \} \]

\[ \text{MM9} = \{ X_{\text{MM9}}, S_{\text{MM9}}, Y_{\text{MM9}}, \delta_{i\text{MM9}}, \delta_{e\text{MM9}}, \lambda_{\text{MM9}}, \text{ta}_{\text{MM9}} \} \]

\[ \text{I}_{\text{MM9}} = \{ \text{MM10} \} \]

\[ Z_{\text{MM9, MM10}} = S_{\text{MM9}} \rightarrow X_{\text{MM10}} \]

\[ \text{CM10} = \{ \text{MM10, I}\text{MM10}, Z_{\text{MM10, } \beta} \} \text{ where } \{ \beta \mid \beta \in \text{I}_{\text{MM10}} \} \]

\[ \text{MM10} = \{ X_{\text{MM10}}, S_{\text{MM10}}, Y_{\text{MM10}}, \delta_{i\text{MM10}}, \delta_{e\text{MM10}}, \lambda_{\text{MM10}}, \text{ta}_{\text{MM10}} \} \]

\[ \text{I}_{\text{MM10}} = \{ \text{MM9} \} \]

\[ Z_{\text{MM10, MM9}} = S_{\text{MM10}} \rightarrow X_{\text{MM9}} \]

\[ \text{ICM4} = \{ \text{CM2} \} \]

\[ Z_{\text{CM4, CM2}} = S_{\text{CM4}} \rightarrow X_{\text{CM2}} \]

\[ \text{CMMA} = \{ \text{MMA, I}\text{MMA}, Z_{\text{MMA, } \beta} \} \text{ where } \{ \beta \mid \beta \in \text{I}_{\text{MMA}} \} \]

\[ \text{MMA} = \{ X_{\text{MMA}}, S_{\text{MMA}}, Y_{\text{MMA}}, \delta_{i\text{MMA}}, \delta_{e\text{MMA}}, \lambda_{\text{MMA}}, \text{ta}_{\text{MMA}} \} \]

\[ \text{I}_{\text{MMA}} = \{ \text{CM2, CM3} \} \]

\[ Z_{\text{MMA, CM2}} = S_{\text{MMA}} \rightarrow X_{\text{CM2}} \]

\[ Z_{\text{MMA, CM3}} = S_{\text{MMA}} \rightarrow X_{\text{CM3}} \]

The Select, Select1, Select2, Select21, Select3, Select4 are the rules or algorithms that decide which manipulation module to be activated for the next event.

The parameters used in this model correspond to the definitions given in chapter 3.
6.4 Simulation of EDS

The simulation model of EDS primarily consists of eleven atomic and 5 coupled simulators. In addition, there are some other general purpose routines that graphically animate the EDS models. Each atomic simulator corresponds to a manipulation module in the system while a coupled simulator represents a coupled model of the manipulation modules.

An atomic simulator of a manipulation module contains a set of instructions that describes the behaviour of the manipulation module according to the functions defined in its mathematical model. A coupled simulator of a module contains information on how the atomic models in the coupled model interact with each other. In addition, it describes the interaction and communication of the module with other modules in the system. All these simulators are assumed to operate concurrently. They were implemented on Simscript and expressed as processes.

At the start of the simulation the following parameters are set:

- The number of plates in the inlet stack
- The linear and angular velocity of the manipulation modules
- The scaling between the simulation time and the real time

The complete list of the source code of EDS simulation can be found in appendix C. The states of the system at some instants of time during the simulation process are, however, illustrated by the diagrams described below.

- Figure 6.3 shows the start up screen where different system parameters can be set.
- The EDS simulation stage set up, based on the selection of parameters chosen in figure 6.3, is illustrated in figure 6.4.
- The state of the system at the instant when the first plate has reached the working area and completely drilled is shown in figure 6.5. These holes, as explained earlier,
are of two different sizes. The drill bit is changed by CM4 after the holes of the first size have been done. The drill bit is changed back to the first one when the second size holes are drilled to prepare the system for the second round of drilling.

- Figure 6.6 illustrates the instant when a new plate is entering into the system from the inlet stack and the completed plate is placed on the conveyor belt.
- The moment when the first completed plate is moved into the storehouse via the conveyor belt is shown in figure 6.7. At the same time, the next plate is moved to the work area where the holes to be drilled have also been marked on.
- Finally, figure 6.8 illustrates the last instant of EDS simulation when all the plates have been drilled and arrived at the storehouse.

**Figure 6.3: Setting of Parameters**

**Figure 6.4: starting of EDS Simulation**
Figure 6.5: First Plate drilled

Figure 6.6: Plates on Conveyor Belt

Figure 6.7: First Plate in the Store

Figure 6.8: All Plates Drilled
The simulation approach used for DME expresses the overall process into a sequence of operations and organises it into elementary operations of manipulation modules. These operations are then translated into motion commands for manipulation modules. This methodology, as obvious from the simulation results, provides a detailed and clear picture of when and what is happening at the shop floor. This is of great help in evaluating the overall design of a system. Most of the existing flexible manufacturing system (FMS) CAD tools do not facilitate simulation of the overall automation system in order to evaluate the proposed FMS design [Chi91]. Rather, they only facilitate the off-line input of the robot's program and subsequent testing of the program by graphic animation of robot's motion in a geometric model of the work scene.

The simulation methodology developed for DME provides a functional model of the manipulation desired to be applied in a production line. This approach, with its mathematical foundation, is a systematic and generic tool applicable to any type of DME systems.

6.5 EDS Event-Based Controller

In order to develop an event-based controller for EDS, each manipulation module in the system is assumed to be outfitted with an appropriate finite-states threshold-type sensor. These sensors, as discussed in chapter 5, divide the state space into partitioning blocks and provide signals to the controller within a definite time window to confirm that the respective manipulation modules have reached the required partitioning blocks.

The event-based controller of EDS is expressed in terms of control models of CM1, CM2, CM3, CM4 and MMA. The control models of CM1, CM2, CM3 and CM4 are obtained by the coupling of the control models of their component modules. All these control models are given in appendix G.
6.5.1 Simulation of Event-Based Controller of EDS

The event-based control models defined and developed in the above sub-sections were simulated using simscript. For the sake of simplicity and in order to accommodate the control signals on the screen, the following assumptions were made for the selected speed of the manipulation modules:

- MM2, MM3 and MM7 normally take 2 units of time to transfer a plate to and from the conveyor.
- Both MM1 and MM8 normally take 1 unit of time to pick or place a plate on the conveyor belt.
- Conveyor belt normally takes 2 units of time to move a plate from one destination to another.
- MM3 takes 1/4 unit of time to move a plate by a distance of \( \frac{\text{plate_height}}{(Y_{\text{max}} + 1)} \). The variable \( Y_{\text{max}} \) is the maximum of the y-coordinates of the locations of all the holes to be drilled. Similarly MM4 takes 1/4 unit of time to move the plate by a distance of \( \frac{\text{plate_width}}{(X_{\text{max}} + 1)} \).
- MM5 takes 1/8 unit of time in locate the drill bit on the plate while MM6 takes 1/4 unit of time to drill a hole.
- Each of MM9 and MM10 normally takes 1 unit of time to change the drill bit.

It was also assumed that all the time windows will have a width of \( \pm 10\% \) of its normal time required to cross a partitioning block.

The simulation results of the controller are shown in figures 6.9 through 6.15. Figure 6.9 represents the controller's action during the normal operation at the top level of the system hierarchy, i.e., in terms of control signals of CM1, CM2, CM3, CM4 and MMA. The controller will stop working by not issuing further control signals in case of any abnormality which is detected by the violation of the time windows. It will then issue an appropriate error message that will help in system diagnostics. A typical malfunctioning
situation is illustrated in figure 6.10 in which CM2 develops some problems and its sensor's response is received after the time window.

Figure 6.9: Normal Controller Operation

Figure 6.10: CM2 Malfunctions

Figures 6.11 through 6.15 show the generation of the control signals of CM1, CM2, CM21, CM3 and CM4 respectively.
Figure 6.11: Control Signal of CM1

Figure 6.12: Control Signal of CM2

Figure 6.13: Control Signal of CM21

Figure 6.14: Control Signal of CM3
The complete code for the simulation of event-based controller of EDS is provided in appendix D. The number of plates to be drilled is provided by the user at the beginning of the simulation.

The event-based controller should be sufficiently fast to ensure that events happen within allowed tolerance of their scheduled time. The processing speed, thus, may be a significant bottleneck especially for hard dead-line applications.

The simulation results appear to be well applicable to real-time systems. The final design, however, may need some fine tuning to account for the difference between the nominal and actual response times of manipulation modules, sensors and other devices.

### 6.6 Summary

Modelling, simulation and event-based control strategy developed for DME in this work was applied to the Extended Drilling System that includes all the basic and essential features of a real manufacturing system. The models developed were successfully
simulated and the simulation results were illustrated by a number of figures. The methodology developed and used provide a generic and systematic tool for developing a DME system.
7.1 Introduction

The focus of this project has been on developing a methodology for modelling and simulation of Distributed Manipulation Environment. This goal has been achieved by considering DME as discrete event system and modelling its behaviour and control accordingly. The previous chapters of this thesis reported of the work conducted.

This concluding chapter attempts to summarise the results obtained. On the basis of these results, some conclusions will be drawn and a few directions for pursuing further related work will be proposed.

7.2 Discrete Event Modelling of DME

According to definition, a DME is a concurrent and modular system in which manipulation modules operate asynchronously to carry out the required task. In order to develop a systematic method to analyse and design a DME and ultimately to control its operation a mathematical model defining the behaviour of DME is required. The DME behaviour is in essence discrete and cannot be modelled using conventional continuous-time methods.
Chapter 7 Conclusion and Further Research

The discrete event modelling methodology employed in this work to model DME matches closely the characteristics of this system particularly reflecting its concurrent and hierarchical structure and discrete nature. The modelling procedure developed is generic and provides a unified framework for static and dynamic behaviour analysis of any type of DME system. This methodology describes the system under study in terms of system states and events; representing the states of manipulation modules at different time intervals and the transitions from one state to another as the result of events.

The developed methodology systematically models the whole DME system according to its physical and logical components considering both atomic and coupled levels. The atomic model describes the behaviour of an individual manipulation module. The coupled model, on the other hand, defines the interaction of an ensemble of manipulation modules operating together to perform a specific task.

7.3 Simulation of Discrete Event Model

The Discrete Event simulation of DME was the second phase of the project. The main focus in this respect was to develop a generic methodology to systematically define the simulation models for DME based on the Discrete Event model developed for it.

The simulation model developed consists of an integrated network of concurrent, communicating and asynchronous processes representing the behaviour of the manipulation modules as defined by their discrete models. The simulation model also addresses the computational and communicational aspects of the manipulation modules. The functions and parameters defined in the simulation model are identical to the discrete event mathematical model of the system.

The simulation model also has exactly the same structure as the mathematical model of the system. It is also decomposed into atomic and coupled simulators representing the behaviour of the atomic and the coupled models respectively. A coupled simulator, likewise, consists of a number of component simulators that may be atomic in nature or
again a coupled simulator of some other component simulators. Both the atomic and the coupled simulators are implemented as processes in the simulation algorithm.

The methodology developed to build simulation models is again generic and can be applied to DME systems of any type or configuration. This is clearly evident from the simulation model of the case study as all the prismatic manipulation modules use the same code defining the behaviour of a typical prismatic manipulation module. This is also true for the revolute modules. The modification and expansion of an existing DME system will thus be more efficient. It will also make the design task easier and remove a great load from the shoulders of the system designer.

7.4 DME Event-Based Control

The development of the event-based controller for DME was an important aspect of this work. This type of control is one level above conventional control approach which enhances the operation of the individual actuators in DME. The major role of this layer of control is to provide a systematic method for sequencing and scheduling the operation of the actuators.

The event-based controller works in an expectation-driven manner. It receives information related to commands and expected response time and windows from the system model. The controller then issues commands to the system under control to move it from one state to another as long as it receives the proper response signals.

An event-based controller primarily contains concurrent, self contained and loosely coupled control models that are derived from the discrete event model of the system. The controller itself can then be expressed as a discrete event model which may be tested, validated and finely tuned through computer simulation prior to its real-time implementation.

The error messages generated by the event-based controller in the events of malfunctioning of one of the components provide important information for diagnostic
purposes. It will greatly reduce the time and efforts to bring the system back to the normal operating mode.

7.5 Further Work

The mathematical modelling developed for DME in this work will pave the way for a more systematic work on this concept. The work conducted in this project can be considered of a preliminary nature towards this end.

The models developed in this work were validated through computer simulation and the results were found very encouraging. The real-time validation of the developed algorithms and logic was not possible at the moment due to unavailability of a system appropriate for such experimentation. The Programmable Array Manipulator (PAM), developed at the University of Wollongong, is an intelligent, concurrent, modular and programmable system [Ciu93], [Won93], [Sha94]. Once PAM is fully operational, it can serve as a system for the real-time implementation of the developed methodologies.

The discrete models developed in this work were produced manually. This task is quite time consuming and cumbersome particularly as the size of the system increases. In addition, a great deal of attention and efforts is required to avoid errors. This situation becomes even worse for event-based control models.

A computer-based tool generating discrete models for the system and its control from a given schematic diagram of the system can speed up the process significantly. The simulation models may also be generated in this process and simulated on the computer.

Another possible direction of the work can be to implement the discrete event simulation models and event-based controller on a parallel computing platform such as transputer. The inherent parallelism and distributed nature of discrete event models lend themselves well for such an approach. This will speed up the simulation and control processes and simplifies the code development.
BIBLIOGRAPHY


[Inr90] Inria¹, *Linear Systems in ( Max,+ ) Algebra*, Proc. of 29th Conf. on Decision and Control, Honolulu, pp151-156, (1990)


¹Inria is a collective name of a group composed of M.Akian, G.Cohen, S.Gaubert, R.Nikoukhah and J.P.Quadrant
Bibliography


Bibliography


APPENDIX A

Code for Simulation of Drilling System

preamble.sim

PREAMBLE
NORMALLY MODE IS UNDEFINED
" STATIC DECLARATION OF ATOMIC AND COUPLED MODELS
PROCESSES INCLUDE CM1, " COUPLED MODEL OF MM_C AND MM_D
SELECT, " SELECT MODULE
UPDATE " USED FOR ANIMATION
EVERY MM_A
HAS A XMA
AND A YMA
DEFINE XMA AND YMA AS REAL VARIABLES
EVERY MM_B
HAS A XMB
AND A YMB
DEFINE XMB AND YMB AS REAL VARIABLES
EVERY MM_C
HAS A XMC
AND A YMC
DEFINE XMC AND YMC AS REAL VARIABLES
EVERY MM_D
HAS A DEG
AND A W
DEFINE DEG AS AN INTEGER VARIABLE
DEFINE W AS A REAL VARIABLE
EVERY MV
HAS AN ICON
AND A XMV
AND A YMV
DEFINE ICON AS A POINTER VARIABLE
DEFINE XMV, YMV AS REAL VARIABLES
RESOURCES
EVERY LINE1 HAS A L1.ICN
DEFINE L1.ICN AS A POINTER VARIABLE
EVERY LINE2 HAS A L2.ICN
DEFINE L2.ICN AS A POINTER VARIABLE
EVERY HOLE1 HAS A H1.ICN
DEFINE H1.ICN AS A POINTER VARIABLE
EVERY HOLE2 HAS A H2.ICN
DEFINE H2.ICN AS A POINTER VARIABLE
" GLOBAL VARIABLES
DEFINE MA, MB, MC, MD, PL AS POINTER VARIABLES
DEFINE H AND V AS AN INTEGER, 1-DIMENSIONAL ARRAY
DEFINE NUM.EVENTS AND I AS AN INTEGER VARIABLES
DEFINE H.MAX, Y.MAX AND DEG AS INTEGER VARIABLES
DEFINE SPEED, DISTANCE AND .TIME AS REAL VARIABLE
DEFINE X_STEP AND Y_STEP AS REAL VARIABLES
DEFINE CLOCKTIME AS DOUBLE VARIABLE
" GRAPHIC VARIABLES
DISPLAY VARIABLES INCLUDE CLOCKTIME
GRAPHIC ENTITIES INCLUDE HEADING AND SHAPE
DYNAMIC GRAPHIC ENTITIES INCLUDE PLATE, HOLE, LINE, SHAPE1
DEFINE .SECONDS TO MEAN UNITS
DEFINE .W TO MEAN 0.02 " GAP BET. TWO ROTATIONS OF MM_D
DEFINE .L TO MEAN 200.0 " LENGTH OF PLATE
DEFINE .H TO MEAN 200.0 " HEIGHT OF PLATE
DEFINE .D TO MEAN 100.0 " DIAGONALITY OF PLATE
END " END OF PREAMBLE

main.sim
MAIN
DEFINE FORM AS A POINTER VARIABLE
DEFINE MAX AS AN INTEGER VARIABLE
" INITIALIZE PARAMETERS
LET MAX = 0
LET H.MAX = 0
LET V.MAX = 0
" GET EVENTS FROM "h.dat"
OPEN 1 FOR INPUT, NAME IS "h.dat", NOERROR
USE 1 FOR INPUT
IF ROPENERR.V EQ 0
READ NUM .EVENTS
RESERVE H(*) AS NUM .EVENTS
FOR I = 1 TO NUM .EVENTS
DO
READ H(I)
LET MAX = MAX + H(I)
IF MAX GT H.MAX
LET H.MAX = MAX
ALWAYS
LOOP
ENDIF
CLOSE 1
LET MAX = 0
" GET EVENTS FROM "v.dat"
OPEN 2 FOR INPUT, NAME IS "v.dat", NOERROR
USE 2 FOR INPUT
IF ROPENERR.V EQ 0
READ NUM .EVENTS
RESERVE V(*) AS NUM .EVENTS
FOR I = 1 TO NUM .EVENTS
DO
READ V(I)
LET MAX = MAX + V(I)
IF MAX GT V.MAX
LET V.MAX = MAX
ALWAYS
LOOP
ENDIF
CLOSE 2
" GET INITIAL PARAMETERS AND START SIMULATION
SHOW FORM WITH "data.frm"
IF ACCEPT.F(FORM, 0) EQ "OK"
LET TIMESCALE.V = DDVAL.A(DFIELD.F("TIMESCALE", FORM))
LET SPEED = DDVAL.A(DFIELD.F("SPEED", FORM))
LET .DEG = DDVAL.A(DFIELD.F("DEG.", FORM))
LET VXFORM.V = 1
LET .TIME = 20.0/SPEED
CALL SETWORLD.R(0.0, 1000.0, 0.0, 1000.0)
CALL INITIALIZE.GRAPHICS
ACTIVATE A SELECT NOW
START SIMULATION
ALWAYS
READ AS /
END  " END OF MAIN

mm_a.sim

PROCESS MM_A GIVEN X, Y
DEFINE X, Y AS REAL VARIABLES
DEFINE DX, DY AS INTEGER VARIABLES
LET DX = X - LOCATION.X(MA)
LET DY = Y - LOCATION.Y(MA)
IF DX NE 0 OR DY NE 0
LET DISTANCE = SQRT.F(DX*DX + DY*DY)
LET SPEED = DISTANCE /.TIME
LET VELOCITY.A(MA) = VELOCITY.F(SPEED,ARCTAN.F(DY,DX))
WORK .TIME .SECONDS
LET VELOCITY.A(MA) = 0
ALWAYS
END " END OF PROCESS MM_A

**mm_b.sim**

PROCESS MM_B GIVEN X,Y
DEFINE X, Y AS REAL VARIABLES
DEFINE DX, DY AS INTEGER VARIABLES
LET DX = X - LOCATION.X(MB)
LET DY = Y - LOCATION.Y(MB)
IF DX NE 0 OR DY NE 0
   LET DISTANCE = SQRT.F(DX*DX + DY*DY)
   LET SPEED = DISTANCE .TIME
   LET VELOCITY.A(MB) = VELOCITY.F(SPEED,ARCTAN.F(DY,DX))
   WORK .TIME .SECONDS
   LET VELOCITY.A(MB) = 0
ALWAYS
END " END OF PROCESS MM_B

**mm_c.sim**

PROCESS MM_C GIVEN X,Y
DEFINE X, Y AS REAL VARIABLES
DEFINE DX, DY AS INTEGER VARIABLES
LET DX = X - LOCATION.X(MC)
LET DY = Y - LOCATION.Y(MC)
IF DX NE 0 OR DY NE 0
   LET DISTANCE = SQRT.F(DX*DX + DY*DY)
   LET SPEED = DISTANCE .TIME
   LET VELOCITY.A(MC) = VELOCITY.F(SPEED,ARCTAN.F(DY,DX))
   WORK .TIME .SECONDS
   LET VELOCITY.A(MC) = 0
ALWAYS
END " END OF PROCESS MM_C

**mm_d.sim**

PROCESS MM_D GIVEN DEG, W
DEFINE DEG AS AN INTEGER VARIABLE
DEFINE W AS A REAL VARIABLE
DEFINE A AS AN INTEGER VARIABLE
FOR A = 1 TO (2*180)/ABS.F(DEG)
   DO
      ADD (PI.C/180)*DEG TO ORIENTATION.A(MD)
      DISPLAY MD
      WAIT W .SECONDS
   LOOP
END " END OF PROCESS MM_D

**cm1.sim**

PROCESS CM1
ACTIVATE A MM_C (LOCATION.X(MC)-20.0, LOCATION.Y(MC)) NOW
CALL MVR (MD, LOCATION.X(MD)-20, LOCATION.Y(MD))
WAIT 1 .SECONDS
ACTIVATE A MM_D (.DEG, .W) NOW
Appendix A

```
WAIT (360* W)/.DEG .SECONDS
WAIT 1 .SECONDS
CALL CONFIRM
ACTIVATE A MM_C (LOCATION.X(MC)+20.0, LOCATION.Y(MC)) NOW
CALL MVR (MD, LOCATION.X(MD)+20, LOCATION.Y(MD))
WAIT 1 UNITS
END " END OF PROCESS CM1

routine1.sim
ROUTE INITIALIZE GRAPHICS
DEFINE TITLE AS POINTER VARIABLES
LET TIMESYNC.V = 'CLOCK.UPDATE'
DISPLAY CLOCKTIME WITH "clock.grf"
CREATE A HEADING CALLED TITLE
DISPLAY TITLE WITH "title.icn" at (25.0, 820.0)
END " END OF ROUTINE1

routine2.sim
ROUTE CLOCK.UPDATE GIVEN TIME YIELDING NEWTIME
DEFINE TIME, NEWTIME AS DOUBLE VARIABLE
LET CLOCKTIME = TIME / (24 * 60 * 60)
LET NEWTIME = TIME
RETURN
END " END OF ROUTINE2

select.sim
PROCESS SELECT
DEFINE BASE AS A POINTER VARIABLE
DEFINE R AS A REAL VARIABLE
CREATE A SHAPE CALLED BASE
DISPLAY BASE WITH "mmal.icn" AT (115.0, 450.0)
CREATE A SHAPE1 CALLED MA
DISPLAY MA WITH "mma2.icn" AT (115.0, 450.0)
CREATE A SHAPE CALLED BASE
DISPLAY BASE WITH "mmb1.icn" AT (300.0, 100.0)
CREATE A SHAPE1 CALLED MB
DISPLAY MB WITH "mmb2.icn" AT (300.0, 100.0)
CREATE A SHAPE CALLED BASE
DISPLAY BASE WITH "mmc1.icn" AT (920.0, 500.0)
CREATE A SHAPE1 CALLED MC
DISPLAY MC WITH "mmc2.icn" AT (920.0, 500.0)
CREATE A SHAPE1 CALLED MD
DISPLAY MD WITH "mmd1.icn" AT (820.0, 500.0)
CREATE A PLATE CALLED PL
DISPLAY PL WITH "plate.icn" AT (350.0, 500.0)
WAIT 2 UNITS
LET N.HOLE2 = NUM .EVENTS
CREATE EVERY HOLE2
FOR EACH HOLE2 DO
LET U.HOLE2(LEOLE2) = 1
LOOP
CALL MARKING
FOR I = 1 TO NUM .EVENTS
DO
WAIT 2 UNITS
ACTIVATE A MM_A (LOCATION.X(MA)+(80.0*H(I)/H.MAX), LOCATION.Y(MA)) NOW
LET R = LOCATION.Y(MB)+(77*V(I)/V.MAX)
ACTIVATE A MM_B (LOCATION.X(MB)+(37*V(I)/V.MAX), R) NOW
ACTIVATE A UPDATE NOW
```
WAIT .TIME+1 UNITS
ACTIVATE A CM1 NOW
WAIT 1+(.TIME+1)*2+(360*.W)/.DEG UNITS
LOOP
END " END OF PROCESS SELECT

mark.sim

ROUTINE MARKING
DEFINE X.INITIAL, Y.INITIAL AS REAL VARIABLES
DEFINE X, XI AND Y AS REAL VARIABLES
DEFINE LEGEND AS A POINTER VARIABLE
LET N.HOLE1 = NUM.EVENTS
LET N.LINE1 = V.MAX
LET N.LINE2 = H.MAX
LET X.STEP = (.L/(H.MAX+1))
LET Y.STEP = (.H/(V.MAX+1))
LET X.INITIAL = 350
LET Y.INITIAL = 500
LET X1 = .D/(V.MAX+1)
CREATE EVERY LINE1
FOR EACH LINE1 DO
  LET U.LINE1(LINE1) = 1
  CREATE A LINE CALLED L1.ICN(LINE1)
  LET Y = Y.INITIAL - (Y.STEP)*(LINE1)
  LET X = X.INITIAL - X1*(LINE1)
  DISPLAY L1.ICN(LINE1) WITH "hline.icn" AT (X,Y)
LOOP
CREATE EVERY LINE2
FOR EACH LINE2 DO
  LET U.LINE2(LINE2) = 1
  CREATE A LINE CALLED L2.ICN(LINE2)
  LET X = X.INITIAL + X.STEP*(LINE2)
  DISPLAY L2.ICN(LINE2) WITH "vline.icn" AT (X,Y.INITIAL)
LOOP
CREATE EVERY HOLE1
FOR EACH HOLE1 DO
  LET U.HOLE1(HOLE1) = 1
  CREATE A HOLE CALLED H1.ICN(HOLE1)
  LET X = X.INITIAL + (X.STEP)*H(HOLE1) - X1*V(HOLE1)
  LET Y = Y.INITIAL - (Y.STEP)*V(HOLE1)
  DISPLAY H1.ICN(HOLE1) WITH "hole1.icn" AT (X,Y)
  LET X.INITIAL = X
  LET Y.INITIAL = Y
LOOP
CREATE A HEADING CALLED LEGEND
DISPLAY LEGEND WITH "legend.icn" AT (10,170)
END " END OF MARK

mv.sim

PROCESS MV GIVEN ICON, X, Y
DEFINE ICON AS A POINTER VARIABLE
DEFINE X, Y AS REAL VARIABLES
DEFINE DX, DY AS INTEGER VARIABLES
LET DX = X - LOCATION.X(ICON)
LET DY = Y - LOCATION.Y(ICON)
IF DX NE 0 OR DY NE 0
  LET DISTANCE = SQRT.F(DX*DX + DY*DY)
  LET SPEED = DISTANCE /.TIME
  LET VELOCITY.A(ICON) = VELOCITY.F(SPEED, ARCTAN.F(DY,DX))
  WORK .TIME .SECONDS
  LET VELOCITY.A(ICON) = 0
ALWAYS
**Appendix A**

END  " END OF MV

**mvr.sim**

ROUTINE MVR GIVEN ICON, X, Y
DEFINE ICON AS A POINTER VARIABLE
DEFINE X, Y AS REAL VARIABLES
DEFINE DX, DY AS INTEGER VARIABLES
LET DX = X - LOCATION.X(ICON)
LET DY = Y - LOCATION.Y(ICON)
IF DX NE 0 OR DY NE 0
    LET DISTANCE = SQRT.F(DX*DX + DY*DY)
    LET SPEED = DISTANCE / TIME
    LET VELOCITY.A(ICON) = VELOCITY.F(SPEED, ARCTAN.F(DY, DX))
    WORK .TIME .SECONDS
    LET VELOCITY.A(ICON) = 0
    ALWAYS
    RETURN
END  " END OF MVR

**confirm.sim**

ROUTINE CONFIRM
DEFINE XH AND YH AS REAL VARIABLES
LET XH = LOCATION.X(H1.ICN(I))
LET YH = LOCATION.Y(H1.ICN(I))
CREATE A HOLE CALLED H2.ICN(I)
DISPLAY H2.ICN(I) WITH "hole2.icn" AT (XH, YH)

END  " END OF CONFIRM

**update.sim**

PROCESS UPDATE
DEFINE A AS AN INTEGER VARIABLE
DEFINE LX AND LY AS REAL VARIABLES
FOR EACH LINE1 DO
    LET LX = LOCATION.X(L1.ICN(LINE1)) + X.STEP*H(I)
    LET LY = LOCATION.Y(L1.ICN(LINE1)) + Y.STEP*V(I)
    ACTIVATE A MV (L1.ICN(LINE1), LX, LY) NOW
LOOP
FOR EACH LINE2 DO
    LET LX = LOCATION.X(L2.ICN(LINE2)) + X.STEP*H(I)
    LET LY = LOCATION.Y(L2.ICN(LINE2)) + Y.STEP*V(I)
    ACTIVATE A MV (L2.ICN(LINE2), LX, LY) NOW
LOOP
FOR EACH HOLE1 DO
    LET LX = LOCATION.X(H1.ICN(HOLE1)) + X.STEP*H(I)
    LET LY = LOCATION.Y(H1.ICN(HOLE1)) + Y.STEP*V(I)
    ACTIVATE A MV (H1.ICN(HOLE1), LX, LY) NOW
LOOP
IF I > 1
    FOR A = 1 TO I-1
        LET LX = LOCATION.X(H2.ICN(A)) + X.STEP*H(I)
        LET LY = LOCATION.Y(H2.ICN(A)) + Y.STEP*V(I)
        ACTIVATE A MV (H2.ICN(A), LX, LY) NOW
        LOOP
        ALWAYS
        LET LX = LOCATION.X(PL) + X.STEP*H(I)
        LET LY = LOCATION.Y(PL) + Y.STEP*V(I)
        ACTIVATE A MV (PL, LX, LY) NOW
END " END OF UPDATE

\textit{h.dat}

6
3
-2
3
-2
-1
4

\textit{v.dat}

6
1
1
1
1
1
0
APPENDIX B

B.1 Code for Event-Based Controller of Drilling System

preamble.sim

PREAMBLE
PROCESSES INCLUDE CMA, CMB AND CMC1
DEFINE MA, MB, MC1 AS POINTER VARIABLES
DEFINE ERROR, LABEL, REPORT AS POINTER VARIABLES
DEFINE A1 AND A2 AS INTEGER VARIABLES
DEFINE B1 AND B2 AS INTEGER VARIABLES
DEFINE C1 AND C2 AS INTEGER VARIABLES
DEFINE X, Y1, Y2, Y3 AS REAL VARIABLES
"SENSOR SIGNALS
DEFINE GO1, GO2, UPDATEA, UPDATEB AS INTEGER VARIABLES
"NORMAL ACTIVE TIME OF MODULES USED FOR TIME WINDOWS
DEFINE MA.S1 TO MEAN 3
DEFINE MA.S2 TO MEAN 2
DEFINE MA.S3 TO MEAN 3
DEFINE MB.S1 TO MEAN 1
DEFINE MB.S2 TO MEAN 1
DEFINE MB.S3 TO MEAN 1
DEFINE CM.S1 TO MEAN 3
DEFINE CM.S2 TO MEAN 3
DEFINE CM.S3 TO MEAN 3
GRAPHIC ENTRIES INCLUDE STATE AND MESSAGE
DEFINE L TO MEAN 33.333
DEFINE H TO MEAN 33.333
END " END OF PREAMBLE

main.sim

MAIN
LET GO1 = 0
LET GO2 = 0
LET UPDATEA = 0
LET UPDATEB = 0
LET X = 150
LET Y1 = 800
LET Y2 = 600
LET Y3 = 400
LET VXFORM.V = 1
LET TIMESCALE.V = 100
CALL SETWORLD.R(0.0, 1000.0, 0.0, 1000.0)
CREATE A MESSAGE CALLED REPORT
DISPLAY REPORT WITH "title.icn" AT (501,900)
CREATE A MESSAGE CALLED REPORT
DISPLAY REPORT WITH "legend.icn" AT (20,275)
ACTIVATE A CMA NOW
ACTIVATE A CMB NOW
ACTIVATE A CMC1 NOW
START SIMULATION
CREATE A MESSAGE CALLED REPORT
DISPLAY REPORT WITH "noerror.icn" AT (625,275)
READ AS /
END " END OF MAIN

cma.sim
PROCESS CMA
DEFINE U AS A REAL VARIABLE
LET A1 = TIME.V
CREATE A MESSAGE CALLED LABEL
DISPLAY LABEL WITH "namea.icn" AT (50, 760)
DEFINE XA, YA AS REAL VARIABLES
LET U = MA.S1
CREATE A STATE CALLED MA
DISPLAY MA WITH "win3d.icn" AT (X+U*.L, Y1)
CREATE A STATE CALLED MA
DISPLAY MA WITH "s3.icn" AT (X, Y1)
CREATE A STATE CALLED MA
DISPLAY MA WITH "sd.icn" AT (X+U*.L, Y1)
WORK U UNITS
LET A2 = TIME.V
IF (A2-A1) <= (U+U/10) AND (A2-A1) >= (U-U/10) " TIME WINDOW
    LET GO1 = 1
ELSE
    IF(A2-A1) < (U-U/10)
        CREATE A MESSAGE CALLED ERROR
        DISPLAY ERROR WITH "errorla.icn" AT (625,275)
        READ AS /
        STOP
    ALWAYS
    IF(A2-A1) > (U+U/10)
        CREATE A MESSAGE CALLED ERROR
        DISPLAY ERROR WITH "errorlb.icn" AT (625,275)
        READ AS /
        STOP
    ALWAYS
    ALWAYS
    LET U = CM.S1
    LET XA = LOCATION.X(MA)
    LET YA = LOCATION.Y(MA)
    CREATE A STATE CALLED MA
    DISPLAY MA WITH "win2d.icn" AT (XA+U*.L, YA-.H)
    CREATE A STATE CALLED MA
    DISPLAY MA WITH "s2.icn" AT (XA, YA-.H)
    CREATE A STATE CALLED MA
    DISPLAY MA WITH "su.icn" AT (XA+U*.L, YA-.H)
    LET A1 = TIME.V
    UNTIL UPDATEA = 1
    DO
        WAIT 1 UNIT
    LOOP
    LET A2 = TIME.V
    IF (A2-A1) <= (U+U/10)+1 AND (A2-A1) >= (U-U/10)+1 " TIME WINDOW
        LET UPDATEA = 0
    ELSE
        IF(A2-A1) < (U-U/10)+1 OR (A2-A1) > (U+U/10)+1
            CREATE A MESSAGE CALLED ERROR
            DISPLAY ERROR WITH "error1a.icn" AT (625,275)
            READ AS /
            STOP
        ALWAYS
        ALWAYS
        LET XA = LOCATION.X(MA)
        LET YA = LOCATION.Y(MA)
        LET U = MA.S2
        LET A1 = TIME.V
        CREATE A STATE CALLED MA
        DISPLAY MA WITH "win2u.icn" AT (XA+U*.L, YA-.H)
        CREATE A STATE CALLED MA
        DISPLAY MA WITH "s2.icn" AT (XA, YA-.H)
        CREATE A STATE CALLED MA
        DISPLAY MA WITH "su.icn" AT (XA+U*.L, YA-.H)
        WORK U-1 UNITS
        WORK 1 UNITS "DELETE THIS LINE TO INTRODUCE WINDOW TIME VIOLATION
OF FIGURE 5.10

LET $A_2 = \text{TIME.V}$

IF $(A_2 - A_1) \leq (U + U/10)$ AND $(A_2 - A_1) \geq (U - U/10)$ "TIME WINDOW"

LET $G_0 = 1$

ELSE

IF $(A_2 - A_1) < (U - U/10)$

CREATE A MESSAGE CALLED ERROR

DISPLAY ERROR WITH "error1a.icn" AT (625,275)

READ AS

STOP

ALWAYS

IF $(A_2 - A_1) > (U + U/10)$

CREATE A MESSAGE CALLED ERROR

DISPLAY ERROR WITH "error1b.icn" AT (625,275)

READ AS

STOP

ALWAYS

ALWAYS

LET $X_A = \text{LOCATION.X(MA)}$

LET $Y_A = \text{LOCATION.Y(MA)}$

LET $U = .CM.S2$

CREATE A STATE CALLED MA

DISPLAY MA WITH "win3u.icn" AT ($X_A + U* L, $Y_A + H)

CREATE A STATE CALLED MA

DISPLAY MA WITH "s3.icn" AT ($X_A, $Y_A + H)

CREATE A STATE CALLED MA

DISPLAY MA WITH "su.icn" AT ($X_A + U* L, $Y_A + H)

LET $A_1 = \text{TIME.V}$

UNTIL $UPDATEA = 1$

DO

WAIT 1 UNIT

LOOP

LET $A_2 = \text{TIME.V}$

IF $(A_2 - A_1) \leq (U + U/10)+1$ AND $(A_2 - A_1) \geq (U - U/10)+1$ "TIME WINDOW"

LET $UPDATEA = 0$

ELSE

IF $(A_2 - A_1) < (U - U/10)+1$ OR $(A_2 - A_1) > (U + U/10)+1$

CREATE A MESSAGE CALLED ERROR

DISPLAY ERROR WITH "errorc1.icn" AT (625,275)

READ AS

STOP

ALWAYS

ALWAYS

LET $X_A = \text{LOCATION.X(MA)}$

LET $Y_A = \text{LOCATION.Y(MA)}$

LET $U = .MA.S3$

LET $A_1 = \text{TIME.V}$

CREATE A STATE CALLED MA

DISPLAY MA WITH "win3d.icn" AT ($X_A + U* L, $Y_A + H)

CREATE A STATE CALLED MA

DISPLAY MA WITH "sd.icn" AT ($X_A + U* L, $Y_A + H)

WORK $U$ UNITS

LET $A_2 = \text{TIME.V}$

IF $(A_2 - A_1) \leq (U + U/10)$ AND $(A_2 - A_1) > (U - U/10)$ "TIME WINDOW"

LET $G_0 = 1$

ELSE

IF $(A_2 - A_1) < (U - U/10)$

CREATE A MESSAGE CALLED ERROR

DISPLAY ERROR WITH "error1a.icn" AT (625,275)

READ AS

STOP

ALWAYS

IF $(A_2 - A_1) > (U + U/10)$

CREATE A MESSAGE CALLED ERROR

DISPLAY ERROR WITH "error1b.icn" AT (625,275)

READ AS

STOP
ALWAYS
LET XA = LOCATION.X(MA)
LET YA = LOCATION.Y(MA)
LET U = .CM.S3
CREATE A STATE CALLED MA
DISPLAY MA WITH "win3d.icn" AT (XA+U*.L, YA-.H)
CREATE A STATE CALLED MA
DISPLAY MA WITH "s3.icn" AT (XA, YA-.H)
CREATE A STATE CALLED MA
DISPLAY MA WITH "sd.icn" AT (XA+U*.L, YA-.H)
LET Al = TIME.V
UNTIL UPDATEA = 1
DO
WAIT 1 UNIT
LOOP
LET A2 = TIME.V
LET UPDATEA = 0
ELSE
IF(A2-A1) < (U-U/10) OR (A2-A1) > (U+U/10)+l
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "error1.icn" AT (625,275)
READ AS/
STOP
ALWAYS
END
ALWAYS
ALWAYS
END CMA

**cmb.sim**

PROCESS CMB
DEFINE U AS A REAL VARIABLE
DEFINE XB, YB AS REAL VARIABLES
LET B1 = TIME.V
CREATE A MESSAGE CALLED LABEL
DISPLAY LABEL WITH "nameb.icn" AT (50, 560)
LET U = .MB.S1
CREATE A STATE CALLED MB
DISPLAY MB WITH "winld.icn" AT (X+U*.L, Y2)
CREATE A STATE CALLED MB
DISPLAY MB WITH "sl.icn" AT (X, Y2)
CREATE A STATE CALLED MB
DISPLAY MB WITH "sd.icn" AT (X+U*.L, Y2)
WORK U UNITS
LET B2 = TIME.V
IF (B2-B1) <= (U+U/10) AND (B2-B1) >= (U-U/10) " TIME WINDOW
LET GO2 = 1
ELSE
IF(B2-B1) < (U-U/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "error2a.icn" AT (625,275)
READ AS/
STOP
ALWAYS
IF(B2-B1) > (U+U/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "error2b.icn" AT (625,275)
READ AS/
STOP
ALWAYS
ALWAYS
LET U = .CM.S1+ MA.S1-MB.S1
LET XB = LOCATION.X(MB)
LET YB = LOCATION.Y(MB)
CREATE A STATE CALLED MB
DISPLAY MB WITH "winlu.icn" AT (XB+U*.L, YB-.H)
CREATE A STATE CALLED MB
DISPLAY MB WITH "s5.icn" AT (XB, YB-.H)
CREATE A STATE CALLED MB
DISPLAY MB WITH "su.icn" AT (XB+U*L, YB-.H)
LET B1 = TIME.V
UNTIL UPDATEB = 1
DO
    WAIT 1 UNIT
LOOP
LET B2 = TIME.V
IF (B2-B1) <= (U+U/10)+1 AND (B2-B1) >= (U-U/10)+1 " TIME WINDOW
    LET UPDATEB = 0
ELSE
    IF(B2-B1) < (U-U/10)+1 OR (B2-B1) > (U+U/10)+1
        CREATE A MESSAGE CALLED ERROR
        DISPLAY ERROR WITH "error2.icn" AT (625,275)
        READ AS/
        STOP
    ALWAYS
    LET XB = LOCATION.X(MB)
    LET YB = LOCATION.Y(MB)
    LET U = .MB.S2
    LET B1 = TIME.V
    CREATE A STATE CALLED MB
    DISPLAY MB WITH "winld.icn" AT (XB+U*L, YB+.H)
    CREATE A STATE CALLED MB
    DISPLAY MB WITH "sl.icn" AT (XB, YB+.H)
    CREATE A STATE CALLED MB
    DISPLAY MB WITH "sd.icn" AT (XB+U*L, YB+.H)
    WORK U UNITS
    LET B2 = TIME.V
    IF (B2-B1) <= (U+U/10) AND (B2-B1) >= (U-U/10) " TIME WINDOW
        LET GO2 = 1
    ELSE
        IF(B2-B1) < (U-U/10)
            CREATE A MESSAGE CALLED ERROR
            DISPLAY ERROR WITH "error2a.icn" AT (625, 275)
            READ AS/
            STOP
        ALWAYS
    IF(B2-B1) > (U+U/10)
        CREATE A MESSAGE CALLED ERROR
        DISPLAY ERROR WITH "error2b.icn" AT (625, 275)
        READ AS/
        STOP
    ALWAYS
    ALWAYS
    LET XB = LOCATION.X(MB)
    LET YB = LOCATION.Y(MB)
    LET U = .CM.S2+.MA.S2-.MB.S2
    CREATE A STATE CALLED MB
    DISPLAY MB WITH "winlu.icn" AT (XB+U*L, YB-.H)
    CREATE A STATE CALLED MB
    DISPLAY MB WITH "s4.icn" AT (XB, YB-.H)
    CREATE A STATE CALLED MB
    DISPLAY MB WITH "su.icn" AT (XB+U*L, YB-.H)
    LET B1 = TIME.V
    UNTIL UPDATEB = 1
DO
    WAIT 1 UNIT
LOOP
LET B2 = TIME.V
IF (B2-B1) <= (U+U/10)+1 AND (B2-B1) >= (U-U/10)+1 " TIME WINDOW
    LET UPDATEB = 0
ELSE
    IF(B2-B1) < (U-U/10)+1 OR (B2-B1) > (U+U/10)+1
        CREATE A MESSAGE CALLED ERROR
        DISPLAY ERROR WITH "error2.icn" AT (625,275)
Appendix B
124

READ AS/
STOP

ALWAYS
LET XB = LOCATION.X(MB)
LET YB = LOCATION.Y(MB)
LET U = MB.S3
LET B1 = TIME.V
CREATE A STATE CALLED MB
DISPLAY MB WITH "winld.icn" AT (XB+U*.L, YB+.H)
CREATE A STATE CALLED MB
DISPLAY MB WITH "sl.icn" AT (XB, YB+.H)
CREATE A STATE CALLED MB
DISPLAY MB WITH "sd.icn" AT (XB+U*.L, YB+.H)
WORK U UNITS
LET B2 = TIME.V
IF (B2-B1) <= (U-U/10) AND (B2-B1) >= (U-U/10) " TIME WINDOW
LET GO2 = 1
ELSE
IF(B2-B1) < (U-U/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "error2a.icn" AT (625,275)
READ AS/
STOP
ALWAYS
IF(B2-B1) > (U+U/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "error2b.icn" AT (625,275)
READ AS/
STOP
ALWAYS
ALWAYS
LET XB = LOCATION.X(MB)
LET YB = LOCATION.Y(MB)
LET U = .CM.S3+.MA.S3-.MB.S3
CREATE A STATE CALLED MB
DISPLAY MB WITH "win5u.icn" AT (XB+U*.L, YB-.H)
CREATE A STATE CALLED MB
DISPLAY MB WITH "s5.icn" AT (XB, YB-.H)
CREATE A STATE CALLED MB
DISPLAY MB WITH "su.icn" AT (XB+U*.L, YB-.H)
LET B1 = TIME.V
UNTIL UPDATEB = 1
DO
WAIT 1 UNIT
LOOP
LET B2 = TIME.V
IF (B2-B1) <= (U-U/10)+1 AND (B2-B1) >= (U-U/10)+1 " TIME WINDOW
LET UPDATEB = 0
ELSE
IF(B2-B1) < (U-U/10)+1 OR (B2-B1) > (U+U/10)+1
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "errorc2.icn" AT (625,275)
READ AS/
STOP
ALWAYS
END
" END OF CMB

cmc1.sim

PROCESS CMC1
DEFINE U AS A REAL VARIABLE
DEFINE XC, YC AS REAL VARIABLES
CREATE A MESSAGE CALLED LABEL
DISPLAY LABEL WITH "namec.icn" AT (50, 380)
LET U = .MA.S1
CREATE A STATE CALLED MC1
DISPLAY MC1 WITH "win3u.icn" AT (X+U*.L, Y3)
CREATE A STATE CALLED MC1
DISPLAY MC1 WITH "s3.icn" AT (X, Y3)
CREATE A STATE CALLED MC1
DISPLAY MC1 WITH "su.icn" AT (X+U*.L, Y3)
LET C1 = TIME.V
UNTIL GO1 = 1 AND GO2 = 1
DO
  WAIT 1 UNITS
END DO
LET C2 = TIME.V
IF (C2-C1) < (U-U/10) OR (C2-C1) > (U+U/10)
  CREATE A MESSAGE CALLED ERROR
  DISPLAY ERROR WITH "error3.icn" AT (625, 275)
  READ AS/
  STOP
END IF
ALWAYS
IF (C2-C1) <= (U+U/10) AND (C2-C1) >= (U-U/10) " TIME WINDOW
LET C1 = TIME.V
LET XC = LOCATION.X(MC1)
LET YC = LOCATION.Y(MC1)
LET U = .CM.S1
CREATE A STATE CALLED MC1
DISPLAY MC1 WITH "win2d.icn" AT (XC+U*.L, YC+.H)
CREATE A STATE CALLED MC1
DISPLAY MC1 WITH "s3.icn" AT (XC, YC+.H)
CREATE A STATE CALLED MC1
DISPLAY MC1 WITH "sd.icn" AT (XC+U*.L, YC+.H)
WORK U UNITS
LET C2 = TIME.V
END IF
ALWAYS
IF (C2-C1) <= (U+U/10) AND (C2-C1) >= (U-U/10) " TIME WINDOW
LET GO1 = 0
LET GO2 = 0
LET UPDATEA = 1
LET UPDATEB = 1
ELSE
  IF (C2-C1) < (U-U/10)
    CREATE A MESSAGE CALLED ERROR
    DISPLAY ERROR WITH "error3a.icn" AT (625, 275)
    READ AS/
    STOP
  END IF
  ALWAYS
  IF (C2-C1) > (U+U/10)
    CREATE A MESSAGE CALLED ERROR
    DISPLAY ERROR WITH "error3b.icn" AT (625, 275)
    READ AS/
    STOP
  END IF
  ALWAYS
ALWAYS
LET XC = LOCATION X(MC1)
LET YC = LOCATION Y(MC1)
LET U = .MA.S2
CREATE A STATE CALLED MC1
DISPLAY MC1 WITH "win2u.icn" AT (XC+U*.L, YC-.H)
CREATE A STATE CALLED MC1
DISPLAY MC1 WITH "s2.icn" AT (XC, YC-.H)
CREATE A STATE CALLED MC1
DISPLAY MC1 WITH "su.icn" AT (XC+U*.L, YC-.H)
LET C1 = TIME.V
UNTIL GO1 = 1 AND GO2 = 1
DO
  WAIT 1 UNITS
END DO
LET C2 = TIME.V
IF (C2-C1) < (U-U/10)+1 OR (C2-C1) > (U+U/10)+1
  CREATE A MESSAGE CALLED ERROR
  DISPLAY ERROR WITH "error3.icn" AT (625, 275)
READ AS /
STOP

ALWAYS
IF (C2-C1) <= (U+U/10)+1 AND (C2-C1) >= (U-U/10)+1 " TIME WINDOW
LET C1 = TIME.V
LET XC = LOCATION.X(MCI)
LET YC = LOCATION.Y(MCI)
LET U = .CM.S2
CREATE A STATE CALLED MCI
DISPLAY MCI WITH "win3d.icn" AT (XC+U*.L, YC+.H)
CREATE A STATE CALLED MCI
DISPLAY MCI WITH "s3.icn" AT (XC, YC+.H)
CREATE A STATE CALLED MCI
DISPLAY MCI WITH "sd.icn" AT (XC+U*.L, YC+.H)
WORK U UNITS
"WORK 2 UNITS THIS LINE WAD ADDED FOR WINDOW TIME VIOLATION OF FIGURE 5.11
LET C2 = TIME.V

ALWAYS
IF (C2-C1) <= (U-U/10) AND (C2-C1) >= (U-U/10) " TIME WINDOW
LET GO1 = 0
LET GO2 = 0
LET UPDATEA = 1
LET UPDATEB = 1
ELSE
IF(C2-C1) < (U-U/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "error3a.icn" AT (625, 275)
READ AS /
STOP
ALWAYS
IF(C2-C1) > (U+U/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "error3b.icn" AT (625, 275)
READ AS /
STOP
ALWAYS
LET XC = LOCATION.X(MCI)
LET YC = LOCATION.Y(MCI)
LET U = .MA.S3
CREATE A STATE CALLED MCI
DISPLAY MCI WITH "win3u.icn" AT (XC+U*.L, YC-.H)
CREATE A STATE CALLED MCI
DISPLAY MCI WITH "s3.icn" AT (XC, YC-.H)
CREATE A STATE CALLED MCI
DISPLAY MCI WITH "su.icn" AT (XC+U*.L, YC-.H)
LET C1 = TIME.V
UNTIL GO1 = 1 AND GO2 = 1
DO
WAIT 1 UNITS
LOOP
LET C2 = TIME.V
IF(C2-C1) < (U-U/10)+1 OR (C2-C1) >= (U-U/10)+1
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "errorc3.icn" AT (625,275)
READ AS /
STOP
ALWAYS
IF (C2-C1) <= (U+U/10)+1 AND (C2-C1) >= (U-U/10) " TIME WINDOW
LET XC = LOCATION.X(MCI)
LET YC = LOCATION.Y(MCI)
LET U = .CM.S3
LET C1 = TIME.V
CREATE A STATE CALLED MCI
DISPLAY MCI WITH "win3d.icn" AT (XC+U*.L, YC+.H)
CREATE A STATE CALLED MCI
DISPLAY MCI WITH "s3.icn" AT (XC, YC+.H)
CREATE A STATE CALLED MCI
DISPLAY MCI WITH "sd.icn" AT (XC+U*.L, YC+.H)
WORK U UNITS
LET C2 = TIME.V
ALWAYS
IF (C2-C1) <= (U+U/10) AND (C2-C1) >= (U-U/10) " TIME WINDOW
LET GO1 = 0
LET GO2 = 0
LET UPDATEA = 1
LET UPDATEB = 1
ELSE
IF (C2-C1) < (U-U/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "error3a.icn" AT (625,275)
READ AS /
STOP
ALWAYS
IF (C2-C1) > (U+U/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "error3b.icn" AT (625,275)
READ AS /
STOP
ALWAYS
END " END OF CMCI

B.2 Code for Control Signal of CM1

preamble.sim

PREAMBLE

PROCESSES INCLUDE MMC, MMD, CM1
DEFINE CS AND DS AS INTEGER VARIABLES
DEFINE ERROR, LABEL, REPORT AS POINTER VARIABLES
DEFINE X, Y1 AS REAL VARIABLES
GRAPHIC ENTITIES INCLUDE STATE AND MESSAGE
DEFINE .MC.S TO MEAN 1
DEFINE .MC.W1 TO MEAN 3
DEFINE .MC.W2 TO MEAN 1
DEFINE .MC.W3 TO MEAN 2
DEFINE .MC.W4 TO MEAN 1
DEFINE .MC.W5 TO MEAN 3
DEFINE .MC.W6 TO MEAN 1
DEFINE .MD.S1 TO MEAN 1
DEFINE .MD.W1 TO MEAN 4
DEFINE .MD.W2 TO MEAN 4
DEFINE .MD.W3 TO MEAN 5
DEFINE .MD.W4 TO MEAN 1
DEFINE .CYCLE.TIME TO MEAN 17
DEFINE L TO MEAN 33.333
DEFINE H TO MEAN 33.333
END " END OF PREAMBLE

main.sim

MAIN
LET X = 150
LET Y1 = 700
LET VXFORM.V = 1
LET TIMESCALE.V = 100
CALL SETWORLD.R(0.0, 1000.0, 0.0, 1000.0)
CREATE A MESSAGE CALLED REPORT
DISPLAY REPORT WITH "title.icn" AT (501,950)
CREATE A MESSAGE CALLED REPORT
DISPLAY REPORT WITH "heading.icn" AT (501,875)
CREATE A MESSAGE CALLED REPORT
DISPLAY REPORT WITH "legend.icn" AT (50, 225)

ACTIVATE A MMC NOW
ACTIVATE A MMD NOW
ACTIVATE A CM1 NOW

START SIMULATION
READ AS/

END " END OF MAIN

**mc.sim**

PROCESS MMC
DEFINE C1, C2 AND U AS INTEGER VARIABLES
DEFINE MC AS A POINTER VARIABLE
DEFINE XC, YC AS REAL VARIABLES
CREATE A MESSAGE CALLED LABEL
DISPLAY LABEL WITH "nam ec.icn" AT (50, Y1-150)

LET CS = 0
LET C1 = TIME.V
LET U = .MC.W1
CREATE A STATE CALLED MC
DISPLAY MC WITH "win1u.icn" AT (X+U*.L, Y1-150)
CREATE A STATE CALLED MC
DISPLAY MC WITH "s3.icn" AT (X, Y1-150)
CREATE A STATE CALLED MC
DISPLAY MC WITH "su.icn" AT (X+U*.L, Y1-150)
WAIT U UNITS
LET C2 = TIME.V
IF (C2-C1) <= (U+U/10) AND (C2-C1) >= (U-U/10) " TIME WINDOW
LET CS = 1
LET C1 = TIME.V
LET U = .MC.S
CREATE A STATE CALLED MC
DISPLAY MC WITH "win1d.icn" AT (XC+U*.L, YC+.H)
CREATE A STATE CALLED MC
DISPLAY MC WITH "sl.icn" AT (XC, YC+.H)
CREATE A STATE CALLED MC
DISPLAY MC WITH "sd.icn" AT (XC+U*.L, YC+.H)
WORK U UNITS
LET C2 = TIME.V
ELSE
IF (C2-C1) < (U-U/10) OR (C2-C1) > (U+U/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "errorcc.icn" AT (50,275)
READ AS/
STOP
ALWAYS

IF (C2-C1) <= (U-U/10) AND (C2-C1) >= (U-U/10) " TIME WINDOW
SELECT CS = 0
LET C1 = TIME.V
LET U = .MC.W2
CREATE A STATE CALLED MC
DISPLAY MC WITH "win1u.icn" AT (XC+U*.L, YC-.H)
CREATE A STATE CALLED MC
DISPLAY MC WITH "s1.icn" AT (XC, YC+.H)
CREATE A STATE CALLED MC
DISPLAY MC WITH "su.icn" AT (XC+U*.L, YC-.H)
WAIT U UNITS
LET C2 = TIME.V
ELSE
IF\( (C2-C1) < (U-U/10) \)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "$errorca.icn" AT (50,275)
READ AS /
STOP
ALWAYS

IF\( (C2-C1) > (U+U/10) \)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "$errorcb.icn" AT (50,275)
READ AS /
STOP
ALWAYS

ALWAYS
IF \( (C2-C1) <= (U+U/10) \) AND \( (C2-C1) >= (U-U/10) \) " TIME WINDOW
LET CS = 1
LET C1 = TIME.V
LET U = .MC.S
LET XC = LOCATION.X(MC)
LET YC = LOCATION.Y(MC)
CREATE A STATE CALLED MC
DISPLAY MC WITH "$winld.icn" AT (XC+U*.L, YC+.H)
CREATE A STATE CALLED MC
DISPLAY MC WITH "$s1.icn" AT (XC, YC+.H)
CREATE A STATE CALLED MC
DISPLAY MC WITH "$sd.icn" AT (XC+U*.L, YC+.H)
WORK U UNITS
LET C2 = TIME.V
ELSE

IF \( (C2-C1) <= (U+U/10) \) OR \( (C2-C1) > (U+U/10) \)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "$errorcc.icn" AT (50,275)
READ AS /
STOP
ALWAYS

ALWAYS
IF \( (C2-C1) <= (U+U/10) \) AND \( (C2-C1) >= (U-U/10) \) " TIME WINDOW
LET CS = 0
LET C1 = TIME.V
LET U = .MC.W3
LET XC = LOCATION.X(MC)
LET YC = LOCATION.Y(MC)
CREATE A STATE CALLED MC
DISPLAY MC WITH "$winlu.icn" AT (XC+U*.L, YC-.H)
CREATE A STATE CALLED MC
DISPLAY MC WITH "$s2.icn" AT (XC, YC-.H)
CREATE A STATE CALLED MC
DISPLAY MC WITH "$su.icn" AT (XC+U*.L, YC-.H)
WAIT U UNITS
LET C2 = TIME.V
ELSE

IF\( (C2-C1) < (U-U/10) \)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "$errorca.icn" AT (50,275)
READ AS /
STOP
ALWAYS

IF\( (C2-C1) > (U+U/10) \)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "$errorcb.icn" AT (50,275)
READ AS /
STOP
ALWAYS

ALWAYS
IF \( (C2-C1) <= (U+U/10) \) AND \( (C2-C1) >= (U-U/10) \) " TIME WINDOW
LET CS = 1
LET C1 = TIME.V
LET U = .MC.S
LET XC = LOCATION.X(MC)
LET YC = LOCATION.Y(MC)
LET U = .MC.W5
LET XC = LOCATION.X(MC)
LET YC = LOCATION.Y(MC)
CREATE A STATE CALLED MC
DISPLAY MC WITH "winlu.icn" AT (XC+U*.L, YC-.H)
CREATE A STATE CALLED MC
DISPLAY MC WITH "s3.icn" AT (XC, YC-.H)
CREATE A STATE CALLED MC
DISPLAY MC WITH "su.icn" AT (XC+U*.L, YC-.H)
WAIT U UNITS
LET C2 = TIME.V
ELSE
  IF(C2-C1) < (U-U/10)
    CREATE A MESSAGE CALLED ERROR
    DISPLAY ERROR WITH "errorca.icn" AT (50, 275)
    READ AS /
    STOP
  ALWAYS
  IF(C2-C1) > (U+U/10)
    CREATE A MESSAGE CALLED ERROR
    DISPLAY ERROR WITH "errorcb.icn" AT (50, 275)
    READ AS /
    STOP
  ALWAYS
  ALWAYS
  IF (C2-C1) <= (U+U/10) AND (C2-C1) >= (U-U/10) " TIME WINDOW
    LET CS = 1
    LET C1 = TIME.V
    LET U = .MC.S
    LET XC = LOCATION.X(MC)
    LET YC = LOCATION.Y(MC)
    CREATE A STATE CALLED MC
    DISPLAY MC WITH "winld.icn" AT (XC+U*.L, YC+.H)
    CREATE A STATE CALLED MC
    DISPLAY MC WITH "si.icn" AT (XC, YC+.H)
    CREATE A STATE CALLED MC
    DISPLAY MC WITH "sd.icn" AT (XC+U*.L, YC+.H)
    WORK U UNITS
    LET C2 = TIME.V
  ELSE
    IF(C2-C1) < (U-U/10) OR (C2-C1) > (U+U/10)
      CREATE A MESSAGE CALLED ERROR
      DISPLAY ERROR WITH "errorcc.icn" AT (50, 275)
      READ AS /
      STOP
    ALWAYS
  ALWAYS
  ALWAYS
  IF (C2-C1) <= (U+U/10) AND (C2-C1) >= (U-U/10) " TIME WINDOW
    LET CS = 0
    LET C1 = TIME.V
    LET U = .MC.W6
    LET XC = LOCATION.X(MC)
    LET YC = LOCATION.Y(MC)
    CREATE A STATE CALLED MC
    DISPLAY MC WITH "winlu.icn" AT (XC+U*.L, YC-.H)
    CREATE A STATE CALLED MC
    DISPLAY MC WITH "si.icn" AT (XC, YC-.H)
    CREATE A STATE CALLED MC
    DISPLAY MC WITH "su.icn" AT (XC+U*.L, YC-.H)
    WAIT U UNITS
    LET C2 = TIME.V
  ELSE
    IF(C2-C1) < (U-U/10)
      CREATE A MESSAGE CALLED ERROR
      DISPLAY ERROR WITH "errorca.icn" AT (50,275)
      READ AS /
      STOP
    ALWAYS
  ALWAYS
  IF(C2-C1) > (U+U/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "errorcb.icn" AT (50,275)
READ AS /
STOP

ALWAYS
IF (C2-C1) <= (U+U/10) AND (C2-C1) >= (U-U/10) " TIME WINDOW
LET CS = 1
LET C1 = TIME.V
LET U = .MC.S
LET XC = LOCATION.X(MC)
LET YC = LOCATION.Y(MC)
CREATE A STATE CALLED MC
DISPLAY MC WITH "win1d.icn" AT (XC+U*.L, YC+.H)
CREATE A STATE CALLED MC
DISPLAY MC WITH "sl.icn" AT (XC, YC+.H)
CREATE A STATE CALLED MC
DISPLAY MC WITH "sd.icn" AT (XC+U*.L, YC+.H)
WORK U UNITS
LET C2 = TIME.V
ELSE
IF(C2-C1) < (U-U/10) OR (C2-C1) > (U+U/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "errorcc.icn" AT (50,275)
READ AS /
STOP

ALWAYS
IF (C2-C1) <= (U+U/10) AND (C2-C1) >= (U-U/10) " TIME WINDOW
LET CS = 0
ELSE
IF(C2-C1) < (U-U/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "errorca.icn" AT (50,275)
READ AS /
STOP
ALWAYS
IF(C2-C1) > (U+U/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "errorcb.icn" AT (50,275)
READ AS /
STOP
ALWAYS
ALWAYS
END " END OF MC

md.sim

PROCESS MMD
DEFINE D1 AND D2 AS INTEGER VARIABLES
DEFINE MD AS A POINTER VARIABLE
DEFINE XD, YD AS REAL VARIABLES
DEFINE U AS A INTEGER VARIABLE
CREATE A MESSAGE CALLED LABEL
DISPLAY LABEL WITH "named.icn" AT (50, Y1-300)
LET DS = 0
LET D1 = TIME.V
LET U = MD.W1
CREATE A STATE CALLED MD
DISPLAY MD WITH "winlu.icn" AT (X+U*.L, Y1-300)
CREATE A STATE CALLED MD
DISPLAY MD WITH "su.icn" AT (X, Y1-300)
CREATE A STATE CALLED MD
DISPLAY MD WITH "sd.icn" AT (X+U*.L, Y1-300)
WAIT U UNITS
LET D2 = TIME.V
IF (D2-D1) <= (U+U/10) AND (D2-D1) >= (U-U/10) " TIME WINDOW
LET DS = 1
LET D1 = TIME.V
LET U = .MD.S
LET XD = LOCATION.X(MD)
LET YD = LOCATION.Y(MD)
CREATE A STATE CALLED MD
DISPLAY MD WITH "winld.icn" AT (XD+U*.L, YD+.H)
CREATE A STATE CALLED MD
DISPLAY MD WITH "sl.icn" AT (XD, YD+.H)
CREATE A STATE CALLED MD
DISPLAY MD WITH "sd.icn" AT (XD+U*.L, YD+.H)
WORK U UNITS
LET D2 = TIME.V
ELSE
    IF(D2-D1) < (U-U/10) OR (D2-D1) > (U+U/10)
    CREATE A MESSAGE CALLED ERROR
    DISPLAY ERROR WITH "errorcl.icn" AT (50,275)
    READ AS/
    STOP
ALWAYS
IF (D2-D1) <= (U+U/10) AND (D2-D1) >= (U-U/10) " TIME WINDOW
LET DS = 0
LET D1 = TIME.V
LET U = .MD.W2
LET XD = LOCATION.X(MD)
LET YD = LOCATION.Y(MD)
CREATE A STATE CALLED MD
DISPLAY MD WITH "winlu.icn" AT (XD+U*.L, YD-.H)
CREATE A STATE CALLED MD
DISPLAY MD WITH "su.icn" AT (XD, YD-.H)
CREATE A STATE CALLED MD
DISPLAY MD WITH "sd.icn" AT (XD+U*.L, YD-.H)
WAIT U UNITS
LET D2 = TIME.V
ELSE
    IF(D2-D1) < (U-U/10)
    CREATE A MESSAGE CALLED ERROR
    DISPLAY ERROR WITH "erorcc.icn" AT (50,275)
    READ AS/
    STOP
ALWAYS
IF(D2-D1) > (U+U/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "errorcd.icn" AT (50,275)
READ AS/
STOP
ALWAYS
ALWAYS
IF (D2-D1) <= (U+U/10) AND (D2-D1) >= (U-U/10) " TIME WINDOW
LET DS = 1
LET D1 = TIME.V
LET U = .MD.S
LET XD = LOCATION.X(MD)
LET YD = LOCATION.Y(MD)
CREATE A STATE CALLED MD
DISPLAY MD WITH "winld.icn" AT (XD+U*.L, YD+.H)
CREATE A STATE CALLED MD
DISPLAY MD WITH "sl.icn" AT (XD, YD+.H)
CREATE A STATE CALLED MD
DISPLAY MD WITH "sd.icn" AT (XD+U*.L, YD+.H)
WORK U UNITS
LET D2 = TIME.V
ELSE
    IF(D2-D1) < (U-U/10) OR (D2-D1) > (U+U/10)
    CREATE A MESSAGE CALLED ERROR
    DISPLAY ERROR WITH "errorcl.icn" AT (50,275)
READ AS/
STOP

ALWAYS
IF (D2-D1) <= (U-U/10) AND (D2-D1) >= (U-U/10) " TIME WINDOW
LET DS = 0
LET D1 = TIME.V
LET U = MD.W3
LET XD = LOCATION.X(MD)
LET YD = LOCATION.Y(MD)
CREATE A STATE CALLED MD
DISPLAY MD WITH "winlu.icn" AT (XD+U*.L, YD-.H)
CREATE A STATE CALLED MD
DISPLAY MD WITH "s5.icn" AT (XD, YD-.H)
CREATE A STATE CALLED MD
DISPLAY MD WITH "su.icn" AT (XD+U*.L, YD-.H)
WAIT U UNITS
LET D2 = TIME.V
ELSE
IF(D2-D1) < (U-U/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "errorda.icn" AT (50,275)
READ AS/
STOP
ALWAYS
IF(D2-D1) > (U+U/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "errordb.icn" AT (50,275)
READ AS/
STOP
ALWAYS
IF (D2-D1) <= (U+U/10) AND (D2-D1) >= (U-U/10) " TIME WINDOW
LET DS = 1
LET D1 = TIME.V
LET U = MD.S
LET XD = LOCATION.X(MD)
LET YD = LOCATION.Y(MD)
CREATE A STATE CALLED MD
DISPLAY MD WITH "winld.icn" AT (XD+U*.L, YD+.H)
CREATE A STATE CALLED MD
DISPLAY MD WITH "sl.icn" AT (XD, YD+.H)
CREATE A STATE CALLED MD
DISPLAY MD WITH "sd.icn" AT (XD+U*.L, YD+.H)
WORK U UNITS
LET D2 = TIME.V
ELSE
IF(D2-D1) < (U-U/10) OR (D2-D1) > (U+U/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "errorcd.icn" AT (50,275)
READ AS/
STOP
ALWAYS
IF (D2-D1) <= (U+U/10) AND (D2-D1) >= (U-U/10) " TIME WINDOW
LET DS = 0
LET D1 = TIME.V
LET U = MD.4
LET XD = LOCATION.X(MD)
LET YD = LOCATION.Y(MD)
CREATE A STATE CALLED MD
DISPLAY MD WITH "s1.icn" AT (XD, YD-.H)
WAIT U UNITS
LET D2 = TIME.V
ELSE
IF(D2-D1) < (U-U/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "errorda.icn" AT (50,275)
READ AS/
Appendix B

STOP
ALWAYS
IF(D2-D1) > (U+U/10)
   CREATE A MESSAGE CALLED ERROR
   DISPLAY ERROR WITH "errordb.icn" AT (50, 275)
   READ AS /
   STOP
ALWAYS

ALWAYS
IF (D2-D1) <= (U+U/10) AND (D2-D1) >= (U-U/10) " TIME WINDOW
   LET DS = 0
ELSE
   IF(D2-D1) < (U-U/10) OR (D2-D1) > (U+U/10)
      CREATE A MESSAGE CALLED ERROR
      DISPLAY ERROR WITH "errordc.icn" AT (50, 275)
      READ AS /
      STOP
      ALWAYS

ALWAYS

END

process CM1

DEFINE MM AS A POINTER VARIABLE
DEFINE XM, YM AS REAL VARIABLES
LET XM = X
LET YM = Y
CREATE A MESSAGE CALLED LABEL
DISPLAY LABEL WITH "namecml.icn" AT (50, YM)
WHILE TIME.V LT .CYCLE.TIME
DO
UNTIL CS = 1 OR DS = 1
DO
WAIT 1 UNIT
CREATE A STATE CALLED MM
DISPLAY MM WITH "sl.icn" AT (XM, YM)
LET XM = LOCATION.X(MM)+.L
LET YM = LOCATION.Y(MM)
LOOP
CREATE A STATE CALLED MM
DISPLAY MM WITH "winlu.icn" AT (XM, YM)
CREATE A STATE CALLED MM
DISPLAY MM WITH "su.icn" AT (XM, YM)
LET YM = LOCATION.Y(MM)+.H
UNTIL CS = 0 AND DS = 0
DO
WAIT 1 UNIT
CREATE A STATE CALLED MM
DISPLAY MM WITH "sl.icn" AT (XM, YM)
LET XM = LOCATION.X(MM)+.L
LET YM = LOCATION.Y(MM)
LOOP
CREATE A STATE CALLED MM
DISPLAY MM WITH "winld.icn" AT (XM, YM)
CREATE A STATE CALLED MM
DISPLAY MM WITH "sd.icn" AT (XM, YM)
LET YM = LOCATION.Y(MM)-.H
LOOP
END " END OF CM1
APPENDIX C

Code for EDS Simulation

preamble.sim

PREAMBLE
" STATIC DECLARATION OF ATOMIC AND COUPLED MODELS
PROCESSES INCLUDE CM1, CM2, CM21, CM3, CM4,  " COUPLED MODELS
SELECT, "SELECT MODULE
CONVEYOR, UPDATE1, UPDATE2 "UTILITY MODULES
" ATOMIC MODELS
EVERY MM1
HAS A XM1
AND A YM1
DEFINE XM1 AND YM1 AS REAL VARIABLES
EVERY MM2
HAS A XM2
AND A YM2
DEFINE XM2 AND YM2 AS REAL VARIABLES
EVERY MM3
HAS A XM3
AND A YM3
DEFINE XM3 AND YM3 AS REAL VARIABLES
EVERY MM4
HAS A XM4
AND A YM4
DEFINE XM4 AND YM4 AS REAL VARIABLES
EVERY MM5
HAS A XM5
AND A YM5
DEFINE XM5 AND YM5 AS REAL VARIABLES
EVERY MM7
HAS A XM7
AND A YM7
DEFINE XM7 AND YM7 AS REAL VARIABLES
EVERY MM8
HAS A XM8
AND A YM8
DEFINE XM8 AND YM8 AS REAL VARIABLES
EVERY MM9
HAS A XM9
AND A YM9
DEFINE XM9 AND YM9 AS REAL VARIABLES
EVERY MM6
HAS A DEG6
AND A W6
DEFINE DEG6 AS AN INTEGER VARIABLE
DEFINE W6 AS A REAL VARIABLE
EVERY MM10
HAS A DEG10
AND A W10
DEFINE DEG10 AS AN INTEGER VARIABLE
DEFINE W10 AS A REAL VARIABLE
EVERY MMA
HAS A DEGA
AND A WA
DEFINE DEGA AS AN INTEGER VARIABLE
DEFINE WA AS A REAL VARIABLE
EVERY MV
HAS AN ICON
AND A XMV
AND A YMV
DEFINE ICON AS A POINTER VARIABLE

136
APPENDIX C

DEFINE XMV,
YMV AS REAL VARIABLES

RESOURCES
EVERY PLATE HAS A P.ICN
DEFINE P.ICN AS A POINTER VARIABLE
EVERY LINE HAS A L.ICN
DEFINE L.ICN AS A POINTER VARIABLE
EVERY DOT HAS A D.ICN
DEFINE D.ICN AS A POINTER VARIABLE
EVERY LINE1 HAS A L1.ICN
DEFINE L1.ICN AS A POINTER VARIABLE
EVERY LINE2 HAS A L2.ICN
DEFINE L2.ICN AS A POINTER VARIABLE
EVERY HOLE1A HAS A H1A.ICN
DEFINE H1A.ICN AS A POINTER VARIABLE
EVERY HOLE1B HAS A H1B.ICN
DEFINE H1B.ICN AS A POINTER VARIABLE
EVERY HOLE2A HAS A H2A.ICN
DEFINE H2A.ICN AS A POINTER VARIABLE
EVERY HOLE2B HAS A H2B.ICN
DEFINE H2B.ICN AS A POINTER VARIABLE

GRAPHIC ENTITIES INCLUDE SHAPE, BASE
DYNAMIC GRAPHIC ENTITIES INCLUDE SHAPE1, SHAPE2, SHAPE3, HOLE

"GLOBAL VARIABLES
DEFINE I, K, K2, N,
HI.MAX, H2.MAX,
V1.MAX, V2.MAX,
NUM.EVENTS.1, NUM.EVENTS.2,
NUM.DONE, NUM.Plate, DEG,
MAXH, MAXV AS INTEGER VARIABLES

DEFINE DISTANCE, SPEED,
X.STEP, Y.STEP,
R3, .TIME AS REAL VARIABLES

DEFINE M1, M2, M3, M4,
M5, M6, M7, M8,
M9, M10, MA,
P1, P2, P3 AS POINTER VARIABLES

DEFINE H1, H2,
V1, V2 AS INTEGER, 1-DIMENSIONAL ARRAY

DEFINE .SECONDS TO MEAN UNITS
DEFINE .W TO MEAN 0.02 "GAP BET. TWO CONSECTIVE ROTATIONS
DEFINE .G TO MEAN 1.0
DEFINE .B TO MEAN 105.88 " PLATE WIDTH
DEFINE .L TO MEAN 108.63 " PLATE LENGTH
DEFINE .D TO MEAN 53.64 " PLATE DIAGONALITY

END " END OF PREAMBLE

main.sim

MAIN

DEFINE I, MAX1 AND MAX2 AS INTEGER VARIABLES
DEFINE FORM AS A POINTER VARIABLE

*INITIALIZATION OF PARAMETERS
LET HI.MAX = 0
LET V1.MAX = 0
LET H2.MAX = 0
LET V2.MAX = 0
LET MAX1 = 0
LET MAX2 = 0
LET NUM.DONE = 0
LET R3 = 258.0

GET EVENTS FROM "file1.dat"
OPEN 1 FOR INPUT, NAME IS "file1.dat", NOERROR
USE 1 FOR INPUT
IF ROPEERR.V EQ 0
READ NUM.EVENTS.1
RESERVE H1(*) AS NUM.EVENTS.1

READ NUM.EVENTS.2
RESERVE H2(*) AS NUM.EVENTS.2

READ NUM.DONE
RESERVE H3(*) AS NUM.DONE
RESERVE V1(*) AS NUM.EVENTS.1
FOR I = 1 TO NUM.EVENTS.1
DO
READ H1(I), V1(I)
LET MAX1 = MAX1 + H1(I)
LET MAX2 = MAX2 + V1(I)
IF MAX1 GT H1.MAX
    LET H1.MAX = MAX1
ALWAYS
IF MAX2 GT V1.MAX
    LET V1.MAX = MAX2
ALWAYS
LOOP
ENDF
CLOSE 1
* GET EVENTS FROM "file2.dat"
LET MAX1 = 0
LET MAX2 = 0
OPEN 2 FOR INPUT, NAME IS "file2.dat", NOERROR
USE 2 FOR INPUT
IF ROPENERR.V EQ 0
READ NUM.EVENTS.2
RESERVE H2(*) AS NUM.EVENTS.2
RESERVE V2(*) AS NUM.EVENTS.2
FOR I = 1 to NUM.EVENTS.2
DO
READ H2(I), V2(I)
LET MAX1 = MAX1 + H2(I)
LET MAX2 = MAX2 + V2(I)
IF MAX1 GT H2.MAX
    LET H2.MAX = MAX1
ALWAYS
IF MAX2 GT V2.MAX
    LET V2.MAX = MAX2
ALWAYS
LOOP
ENDF
CLOSE 2
* GET INITIAL PARAMETERS, SET UP GRAPHIC SCREEN AND START SIMULATION
SHOW FORM WITH "data.firm"
IF ACCEPT.FORM, 0 EQ "OK"
LET TIMESCALE.V = DDVAL.A(DFIELD.F("TIMESCAIE", FORM))
LET SPEED = DDVAL.A(DFIELD.F("SPEED", FORM))
LET .NUM.PLATE = DDVAL.A(DFIELD.F("NUMBER", FORM))
LET .DEG = DDVAL.A(DFIELD.F("DEG", FORM))
LET VXFORM.V = 1
CALL SETWORLD.R(0.0, 1000.0, 0.0, 1000.0)
LET N.PLATE = .NUM.PLATE
LET N = .NUM.PLATE
LET .TIME = 20.0/SPEED
CALL SETUP1
CALL SETUP2
ACTIVATE A SELECT NOW
START SIMULATION
"READ AS /
ALWAYS
END
"END OF MAIN

cm1.sim

PROCESS CM1 " COUPLED MODEL OF MM1 & MM2
IF NUM.DONE LT .NUM.PLATE
CALL MVR (M1, LOCATION.X(M1)-11.0, LOCATION.Y(M1)-25.0)
WAIT .G.SECONDS
ACTIVATE A MM1 (LOCATION.X(M1)+11.0, LOCATION.Y(M1)+25.0) NOW
CALL MVR (P.ICN(N), LOCATION.X(P.ICN(N)), LOCATION.Y(P.ICN(N))+10.0)
WAIT .G.SECONDS
"READ AS /
ALWAYS
END
Appendix C

139

ACTIVATE A MM2 (LOCATION.X(M2), LOCATION.Y(M2)-50.0) NOW
CALL MVR (P.ICN(N), LOCATION.X(P.ICN(N)), 440.0)
WAIT .G .SECONDS
ACTIVATE A MM1 (LOCATION.X(M1)-11.0, LOCATION.Y(M1)-25.0) NOW
CALL MVR (P.ICN(N), LOCATION.X(P.ICN(N)), LOCATION.Y(P.ICN(N))-10.0)
WAIT .G .SECONDS
CALL MVR (M1, LOCATION.X(M1)+11.0, LOCATION.Y(M1)+25.0) "conveyor
WAIT .G .SECONDS
WAIT .TIME+1 .SECONDS
CALL MVR (M2, LOCATION.X(M2), LOCATION.Y(M2)+50.0)

ALWAYS

END " END OF CM1

cm2.sim

PROCESS CM2 " COUPLED MODEL OF MM3, MM4 AND CM21
DEFINE R1 AND R2 AS INTEGER VARIABLES
DEFINE PX, PY AND L1 AS REAL VARIABLES
LET R1 = 0
LET R2 = 0
IF NUM.DONE >= 1
ACTIVATE A MM3 (LOCATION.X(M3), 710) NOW
FOR EACH HOLE1B DO
LET L1 = LOCATION.Y(H1B.ICN(HOLE1B))-LOCATION.Y(P.ICN(N+1))
ACTIVATE A MV (H1B.ICN(HOLE1B), LOCATION.X(H1B.ICN(HOLE1B)), 430.0+L1) NOW
LOOP
FOR EACH HOLE2B DO
LET L1 = LOCATION.Y(H2B.ICN(HOLE2B))-LOCATION.Y(P.ICN(N+1))
ACTIVATE A MV (H2B.ICN(HOLE2B), LOCATION.X(H2B.ICN(HOLE2B)), 430.0+L1) NOW
LOOP
CALL MVR (P.ICN(N+1), LOCATION.X(P.ICN(N+1)), 430.0)
WAIT .G .SECONDS
ELSE
WAIT .TIME+1 .SECONDS
ALWAYS
WAIT 3*. (TIME+1) .SECONDS
IF NUM.DONE LT .NUM.PLATE
ACTIVATE A MM3 (LOCATION.X(M3), LOCATION.Y(M3)+50.0) NOW
CALL MVR (P.ICN(N), LOCATION.X(P.ICN(N)), 580.0)
CALL MARKING
WAIT .G .SECONDS
FOR K = 1 TO NUM EVENTS.1
DO
LET PX = LOCATION.X(P.ICN(N))-X.STEP*H1(K)
LET PY = LOCATION.Y(P.ICN(N))+(Y.STEP*V1(K))/2 "HALF
ACTIVATE A MM3 (LOCATION.X(M3), LOCATION.Y(M3)+(50*V1(K))/MAXV) NOW
ACTIVATE A MM4 (LOCATION.X(M4)+(-50*H1(K))/MAXH, LOCATION.Y(M4)) NOW
ACTIVATE A UPDATE1 NOW
CALL MVR (P.ICN(N), PX, PY)
WAIT .G .SECONDS
ACTIVATE A CM21 NOW
WAIT (.TIME+1)+((360*.W)/.DEG) .SECONDS
CALL CONFIRM1
WAIT .TIME+1+1 .SECONDS
LET R1 = R1 + H1(K)
LET R2 = R2 + V1(K)
LOOP
WAIT (.TIME+1)+2*((360*.W/.DEG)+1) .SECONDS
FOR K = 1 TO NUM EVENTS.2
DO
IF K = 1
LET H2(K) = H2(K) - R1
LET V2(K) = V2(K) - R2
ALWAYS
LET PX = LOCATION.X(P.ICN(N))-X.STEP*H2(K)
LET PY = LOCATION.Y(P.ICN(N))+(Y.STEP*V2(K))/2 "HALF
ACTIVATE A MM3 (LOCATION.X(M3), LOCATION.Y(M3)+(50*V2(K))/MAXV) NOW
Appendix C

ACTIVATE A MM4 (LOCATION.X(M4)+(-50*H2(K))/MAXH, LOCATION.Y(M4)) NOW
ACTIVATE A UPDATE2 NOW
CALL MVR (P.ICN(N), PX, PY)
WAIT .G .SECONDS
ACTIVATE A CM21 NOW
WAIT (.TIME+1)+((360*W)/.DEG) .SECONDS
CALL CONFIRM2
WAIT .TIME+1+1 .SECONDS
LOOP
ALWAYS
LET H2(1) = H2(1) + R1
LET V2(1) = V2(1) + R2
WAIT 2*(360*W/.DEG)+.G .SECONDS
ACTIVATE A MM4 (125, LOCATION.Y(M4)) NOW
WAIT .TIME+1 .SECONDS
END

" END OF CM2

cm21.sim

PROCESS CM21 " COUPLED MODEL OF MM5 & MM6
ACTIVATE A MM5 (LOCATION.X(M5)-11.0, LOCATION.Y(M5)-25.0) NOW
CALL MVR (M6, LOCATION.X(M6)-11.0, LOCATION.Y(M6)-25.0)
WAIT 1 .SECONDS
ACTIVATE A MM6 (.DEG, .W) NOW
WAIT (360 *.W)/.DEG .SECONDS
WAIT 1 .SECONDS
ACTIVATE A MM5 (LOCATION.X(M5)+11.0, LOCATION.Y(M5)+25.0) NOW
CALL MVR (M6, LOCATION.X(M6)+11.0, LOCATION.Y(M6)+25.0)
WAIT 1 .SECONDS
END

" END OF CM21

cm3.sim

PROCESS CM3 " COUPLED MODEL OF MM7 & MM8
DEFINE LI AS A REAL VARIABLE
IF NUM.DONE >= 1
CALL MVR (M7, LOCATION.X(M7), LOCATION.Y(M7)+50.0)
WAIT .G .SECONDS
WAIT 2*(TIME+1) .SECONDS
ACTIVATE A MM8 (LOCATION.X(M8)+11.0, LOCATION.Y(M8)+25.0) NOW
FOR EACH HOLE1B DO
LET L1 = LOCATION.Y(H1B.ICN(HOLE1B))+10
ACTIVATE A MV (HIB.ICN(HOLE1B), LOCATION.X(H1B.ICN(HOLE1B)), LI) NOW
LOOP
FOR EACH HOLE2B DO
LET L1 = LOCATION.Y(H2B.ICN(HOLE2B))+10
ACTIVATE A MV (H2B.ICN(HOLE2B), LOCATION.X(H2B.ICN(HOLE2B)), LI) NOW
LOOP
CALL MVR (P.ICN(N+1), LOCATION.X(P.ICN(N+1)), LOCATION.Y(P.ICN(N+1))+10)
WAIT .G .SECONDS
ACTIVATE A MV (P.ICN(N+1), LOCATION.X(P.ICN(N+1)), R3) NOW
FOR EACH HOLE1B DO
LET L1 = LOCATION.Y(H1B.ICN(HOLE1B))-LOCATION.Y(P.ICN(N+1))
ACTIVATE A MV (HIB.ICN(HOLE1B), LOCATION.X(H1B.ICN(HOLE1B)), R3+L1) NOW
LOOP
FOR EACH HOLE2B DO
LET L1 = LOCATION.Y(H2B.ICN(HOLE2B))-LOCATION.Y(P.ICN(N+1))
ACTIVATE A MV (H2B.ICN(HOLE2B), LOCATION.X(H2B.ICN(HOLE2B)), R3+L1) NOW
LOOP
CALL MVR (M7, LOCATION.X(M7), (LOCATION.Y(M7)-50.0))
WAIT .G .SECONDS
ACTIVATE A MM8 (LOCATION.X(M8)-11.0, LOCATION.Y(M8)-25.0) NOW
FOR EACH HOLE1B DO
LET L1 = LOCATION.Y(H1B.ICN(HOLE1B))-8
Appendix C

ACTIVATE A MV (H1B.ICN(HOLE1B), LOCATION.X(H1B.ICN(HOLE1B)), L1) NOW
LOOP
FOR EACH HOLE2B DO
LET L1 = LOCATION.Y(H2B.ICN(HOLE2B))-8
ACTIVATE A MV (H2B.ICN(HOLE2B), LOCATION.X(H2B.ICN(HOLE2B)), L1) NOW
LOOP
CALL MVR (P.ICN(N+1), LOCATION.X(P.ICN(N+1)), LOCATION.Y(P.ICN(N+1))-8)
ALWAYS
END " END OF CM3

cm4.sim

PROCESS CM4 " COUPLED MODEL OF MM9 & MM10
ACTIVATE A MM9(LOCATION.X(M9)+50.0, LOCATION.Y(M9)) NOW
CALL MVR (M10, LOCATION.X(M10)+50.0, LOCATION.Y(M10))
WAIT .G .SECONDS
ACTIVATE A MM10 (.DEG., .W) NOW
WAIT (360 *.W)/.DEG .SECONDS
DISPLAY M6 WITH "mm6b.icn" AT (543,800)
WAIT .G .SECONDS
ACTIVATE A MM10 (-.DEG., .W) NOW
WAIT (360 *.W)/.DEG .SECONDS
WAIT .G .SECONDS
WAIT (NUM.EVENTS,2)*(3*(.TIME+1)+(360* .W/.DEG+1)) .SECONDS
ACTIVATE A MM10 (.DEG., .W) NOW
WAIT (360 *.W)/.DEG .SECONDS
DISPLAY M6 WITH "mm6a.icn" AT (545,815)
WAIT .G .SECONDS
ACTIVATE A MM10 (-.DEG., .W) NOW
WAIT (360 *.W)/.DEG .SECONDS
WAIT .G .SECONDS
ACTIVATE A MM9(LOCATION.X(M9)-50.0, LOCATION.Y(M9)) NOW
CALL MVR (M10, LOCATION.X(M10)-50.0, LOCATION.Y(M10))
END " END OF CM4

mm1.sim

PROCESS MM1 GIVEN X, Y " MANIPULATION MODULE 1
DEFINE X, Y AS REAL VARIABLES
DEFINE DX, DY AS INTEGER VARIABLES
LET DX = X - LOCATION.X(M1)
LET DY = Y - LOCATION.Y(M1)
IF DX NE 0 OR DY NE 0
LET DISTANCE = SQRT.F(DX*DX + DY*DY)
LET SPEED = DISTANCE /.TIME
LET VELOCITY.A(M1) = VELOCITY,F(SPEED,ARCTAN.F(DY,DX))
WORK .TIME .SECONDS
LET VELOCITY.A(M1) = 0
ALWAYS
END " END OF MMA

mm2.sim

PROCESS MM2 GIVEN X, Y " MANIPULATION MODULE 2
DEFINE X, Y AS REAL VARIABLES
DEFINE DX, DY AS INTEGER VARIABLES
LET DX = X - LOCATION.X(M2)
LET DY = Y - LOCATION.Y(M2)
IF DX NE 0 OR DY NE 0
LET DISTANCE = SQRT.F(DX*DX + DY*DY)
LET SPEED = DISTANCE / TIME
LET VELOCITY.A(M2) = VELOCITY.F(SPEED, ARCTAN.F(DY, DX))
WORK .TIME .SECONDS
LET VELOCITY.A(M2) = 0
ALWAYS
END " END OF MM2

mm3.sim

PROCESS MM3 GIVEN X, Y " MANIPULATION MODULE 3
DEFINE X, Y AS REAL VARIABLES
DEFINE DX, DY AS INTEGER VARIABLES
LET DX = X - LOCATION.X(M3)
LET DY = Y - LOCATION.Y(M3)
IF DX NE 0 OR DY NE 0
LET DISTANCE = SQRT.F(DX*DX + DY*DY)
LET SPEED = DISTANCE / TIME
LET VELOCITY.A(M3) = VELOCITY.F(SPEED, ARCTAN.F(DY, DX))
WORK .TIME .SECONDS
LET VELOCITY.A(M3) = 0
ALWAYS
END " END OF MM3

mm4.sim

PROCESS MM4 GIVEN X, Y " MANIPULATION MODULE 4
DEFINE X, Y AS REAL VARIABLES
DEFINE DX, DY AS INTEGER VARIABLES
LET DX = X - LOCATION.X(M4)
LET DY = Y - LOCATION.Y(M4)
IF DX NE 0 OR DY NE 0
LET DISTANCE = SQRT.F(DX*DX + DY*DY)
LET SPEED = DISTANCE / TIME
LET VELOCITY.A(M4) = VELOCITY.F(SPEED, ARCTAN.F(DY, DX))
WORK .TIME .SECONDS
LET VELOCITY.A(M4) = 0
ALWAYS
END " END OF MM4

mm5.sim

PROCESS MM5 GIVEN X, Y " MANIPULATION MODULE 5
DEFINE X, Y AS REAL VARIABLES
DEFINE DX, DY AS INTEGER VARIABLES
LET DX = X - LOCATION.X(M5)
LET DY = Y - LOCATION.Y(M5)
IF DX NE 0 OR DY NE 0
LET DISTANCE = SQRT.F(DX*DX + DY*DY)
LET SPEED = DISTANCE / TIME
LET VELOCITY.A(M5) = VELOCITY.F(SPEED, ARCTAN.F(DY, DX))
WORK .TIME .SECONDS
LET VELOCITY.A(M5) = 0
ALWAYS
END " END OF MM5

mm6.sim

PROCESS MM6 GIVEN DEG, W " MANIPULATION MODULE MM6
DEFINE DEG AS AN INTEGER VARIABLE
DEFINE A AS AN INTEGER VARIABLE
DEFINE W AS A REAL VARIABLE
FOR A = 1 TO (2*180)/ABS.F(DEG)
DO
ADD (PI/180)*DEG TO ORIENTATION.A(M6)
DISPLAY M6
WAIT W .SECONDS
LOOP
END " END OF PROCESS MM6

**mm7.sim**

PROCESS MM7 GIVEN X, Y " MANIPULATION MODULE 7
DEFINE X, Y AS REAL VARIABLES
DEFINE DX, DY AS INTEGER VARIABLES
LET DX = X - LOCATION.X(M7)
LET DY = Y - LOCATION.Y(M7)
IF DX NE 0 OR DY NE 0
   LET DISTANCE = SQRT.F(DX*DX + DY*DY)
   LET SPEED = DISTANCE / .TIME
   LET VELOCITY.A(M7) = VELOCITY.F(SPEED, ARCTAN.F(DY,DX))
   WORK .TIME .SECONDS
   LET VELOCITY.A(M7) = 0
ALWAYS
END " END OF MM7

**mm8.sim**

PROCESS MM8 GIVEN X, Y " MANIPULATION MODULE 8
DEFINE X, Y AS REAL VARIABLES
DEFINE DX, DY AS INTEGER VARIABLES
LET DX = X - LOCATION.X(M8)
LET DY = Y - LOCATION.Y(M8)
IF DX NE 0 OR DY NE 0
   LET DISTANCE = SQRT.F(DX*DX + DY*DY)
   LET SPEED = DISTANCE / .TIME
   LET VELOCITY.A(M8) = VELOCITY.F(SPEED, ARCTAN.F(DY,DX))
   WORK .TIME .SECONDS
   LET VELOCITY.A(M8) = 0
ALWAYS
END " END OF MM8

**mm9.sim**

PROCESS MM9 GIVEN X, Y " MANIPULATION MODULE 9
DEFINE X, Y AS REAL VARIABLES
DEFINE DX, DY AS INTEGER VARIABLES
LET DX = X - LOCATION.X(M9)
LET DY = Y - LOCATION.Y(M9)
IF DX NE 0 OR DY NE 0
   LET DISTANCE = SQRT.F(DX*DX + DY*DY)
   LET SPEED = DISTANCE / .TIME
   LET VELOCITY.A(M9) = VELOCITY.F(SPEED, ARCTAN.F(DY,DX))
   WORK .TIME .SECONDS
   LET VELOCITY.A(M9) = 0
ALWAYS
END " END OF MM9

**mm10.sim**

PROCESS MM10 GIVEN DEG, W " MANIPULATION MODULE 10
DEFINE DEG AS AN INTEGER VARIABLE
DEFINE A AS AN INTEGER VARIABLE
DEFINE W AS A REAL VARIABLE
FOR A = 1 TO (2*180)/ABS.F(DEG)
   DO
      ADD (PI.C/180)*DEG TO ORIENTATION.A(M10)
      DISPLAY M10
   WAIT W .SECONDS
LOOP

END " END OF PROCESS MM10

mma.sim

PROCESS MMA GIVEN DEG, W " MANIPULATION MODULE A
DEFINE DEG AS AN INTEGER VARIABLE
DEFINE W AS A REAL VARIABLE
UNTIL S1 = 0
DO
   ADD (PI.C/180)*DEG TO ORIENTATION.A(MA)
   DISPLAY MA
   WAIT W .SECONDS
LOOP

END " END OF PROCESS MMA

setup1.sim

ROUTINE SETUP1 " USED IN CREATING INITIAL SET UP
DEFINE
   X, Y AS AN INTEGER VARIABLE
   STATIC AS A POINTER VARIABLE
LET X = 798
LET Y = 580
CREATE A BASE CALLED STATIC
DISPLAY STATIC WITH "title1.icn" AT (285,70)
CREATE EVERY PLATE
FOR EACH PLATE DO
   LET U.PLATE(PLATE) = 1
   CREATE A SHAPE1 CALLED P.ICN(PLATE)
   DISPLAY P.ICN(PLATE) WITH "p3.icn" AT (X,Y)
   LET Y = Y+10
LOOP
CREATE A BASE CALLED P1
DISPLAY P1 WITH "2.1c" AT (845,635)
CREATE A BASE CALLED P2
DISPLAY P2 WITH "p1.icn" AT (444,580)
CREATE A BASE CALLED P3
DISPLAY P3 WITH "p2.icn" AT (90,258)
CREATE A BASE CALLED STATIC
DISPLAY STATIC WITH "mmla.icn" AT (950,820)
CREATE A SHAPE1 CALLED M1
DISPLAY M1 WITH "mmlb.icn" AT (950,820)
CREATE A BASE CALLED STATIC
DISPLAY STATIC WITH "mm2a.icn" AT (760,760)
CREATE A SHAPE1 CALLED M2
DISPLAY M2 WITH "mmlb.icn" AT (760,760)
CREATE A BASE CALLED STATIC
DISPLAY STATIC WITH "mm3a.icn" AT (125,590)
CREATE A SHAPE1 CALLED M3
DISPLAY M3 WITH "mmlb.icn" AT (125,590)
CREATE A BASE CALLED STATIC
DISPLAY STATIC WITH "mm4a.icn" AT (570,870)
CREATE A SHAPE1 CALLED M4
DISPLAY M4 WITH "mm4b.icn" AT (570,870)
CREATE A BASE CALLED STATIC
DISPLAY STATIC WITH "mm5a.icn" AT (545,815)
CREATE A SHAPE1 CALLED M5
DISPLAY M5 WITH "mm5b.icn" AT (545,815)
CREATE A BASE CALLED STATIC
DISPLAY STATIC WITH "mm6a.icn" AT (310,260)
CREATE A SHAPE1 CALLED M7
DISPLAY M7 WITH “mm7b.icn” AT (310,260)
CREATE A BASE CALLED STATIC
DISPLAY STATIC WITH “mm8a.icn” AT (100,135)
CREATE A SHAPE1 CALLED M8
DISPLAY M8 WITH “mm8b.icn” AT (100,135)
CREATE A BASE CALLED STATIC
DISPLAY STATIC WITH “mm9a.icn” AT (170,840)
CREATE A SHAPE1 CALLED M9
DISPLAY M9 WITH “mm9b.icn” AT (170,840)
CREATE A SHAPE1 CALLED M10
DISPLAY M10 WITH “mm10a.icn” AT (250,840)
CREATE A BASE CALLED STATIC
DISPLAY STATIC WITH “mm1a.icn” AT (780,250)
CREATE A BASE CALLED STATIC
DISPLAY STATIC WITH “mm2a.icn” AT (780,250)
CREATE A SHAPE1 CALLED MA
DISPLAY MA WITH “mm3.icn” AT (780,340)
END

setup2.sim

ROUTINE SETUP2 “USED IN CREATING INITIAL SET UP
DEFINE C AS A POINTER VARIABLE
DEFINE XI, Y1 AS AN INTEGER VARIABLE
LET XI = 50
LET Y1 = 421
LET N.LINE = (1000-2*XI)/XI
LET N.DOT = (1000-2*XI)/XI
CREATE A BASE CALLED C
DISPLAY C WITH “tl.icn” AT (550, 490)
CREATE EVERY LINE
CREATE EVERY DOT
FOR EACH DOT DO
  LET U.DOT(DOT) = 1
  LET U.LINE(LINE) = 1
CREATE A SHAPE CALLED L.ICN(LINE)
DISPLAY L.ICN(LINE) WITH “conv1.icn” AT (XI, Y1)
CREATE A SHAPE2 CALLED D.ICN(DOT)
DISPLAY D.ICN(DOT) WITH “conv2.icn” AT (XI, Y1)
LET XI = XI + 50
LOOP
END

select.sim

PROCESS SELECT “SELECT MODULE
DEFINE J AS AN INTEGER VARIABLE
DEFINE L1 AS REAL VARIABLE
LET L1 = 0.0
LET N.HOLE1B = NUM.EVENTS.1
CREATE EVERY HOLE1B
FOR EACH HOLE1B DO
  LET U.HOLE1B(HOLE1B) = 1
LOOP
LET N.HOLE2B = NUM.EVENTS.2
CREATE EVERY HOLE2B
FOR EACH HOLE2B DO
  LET U.HOLE2B(HOLE2B) = 1
LOOP
WAIT 3*.G .SECONDS
DESTROY A BASE CALLED P1
DESTROY A BASE CALLED P2
DESTROY A BASE CALLED P3
WAIT .G .SECONDS
FOR J=1 TO .NUM.PLATE+1
DO
ACTIVATE A CM1 NOW
WAIT 2*(TIME+1) SECONDS
ACTIVATE A CM2 NOW
ACTIVATE A CM3 NOW
WAIT 2*(TIME+1) SECONDS
LET S1 = 1
ACTIVATE A MMA (10, 0.02) NOW
ACTIVATE A CONVEYOR NOW
IF NUM.DONE LT 1
CALL MVR (P.ICN(N), 444.0, LOCATION.Y(P.ICN(N)))
ALWAYS
IF NUM.DONE >= 1 AND NUM.DONE < NUM.PLATE
ACTIVATE A MV (P.ICN(N), 444.0, LOCATION.Y(P.ICN(N))) NOW
FOR EACH HOLE1B DO
LET L1 = LOCATION.X(H1B.ICN(HOLE1B))-LOCATION.X(P.ICN(N+1))
ACTIVATE A MV (H1B.ICN(HOLE1B), 90.0+L1, LOCATION.Y(H1B.ICN(HOLE1B))) NOW
LOOP
FOR EACH HOLE2B DO
LET L1 = LOCATION.X(H2B.ICN(HOLE2B))-LOCATION.X(P.ICN(N+1))
ACTIVATE A MV (H2B.ICN(HOLE2B), 90.0+L1, LOCATION.Y(H2B.ICN(HOLE2B))) NOW
LOOP
CALL MVR (P.ICN(N+1), 90.0, LOCATION.Y(P.ICN(N+1)))
ALWAYS
IF NUM.DONE > 1 AND NUM.DONE < NUM.PLATE
FOR EACH HOLE1B DO
LET L1 = LOCATION.X(H1B.ICN(HOLE1B))-LOCATION.X(P.ICN(N+1))
ACTIVATE A MV (H1B.ICN(HOLE1B), 90.0+L1, LOCATION.Y(H1B.ICN(HOLE1B))) NOW
LOOP
FOR EACH HOLE2B DO
LET L1 = LOCATION.X(H2B.ICN(HOLE2B))-LOCATION.X(P.ICN(N+1))
ACTIVATE A MV (H2B.ICN(HOLE2B), 90.0+L1, LOCATION.Y(H2B.ICN(HOLE2B))) NOW
LOOP
CALL MVR (P.ICN(N+1), 90.0, LOCATION.Y(P.ICN(N+1)))
ALWAYS
LET S1 = 0
WAIT .G SECONDS
WAIT 2*(TIME+1) SECONDS
IF NUM.DONE LT NUM.PLATE
WAIT (3*(TIME+1)+((360*.W/.DEG)+1))*(NUM.EVENTS.1) SECONDS
ACTIVATE A CM4 NOW
WAIT (TIME+1)+2*((360*.W/.DEG)+1) SECONDS
WAIT (3*(TIME+1)+((360*.W/.DEG)+1))*(NUM.EVENTS.2) SECONDS
WAIT (TIME+1)+2*((360*.W/.DEG)+1) SECONDS
FOR EACH LINE1 DO
DESTROY A SHAPE3 CALLED L1.ICN(LINE1)
LOOP
FOR EACH LINE2 DO
DESTROY A SHAPE3 CALLED L2.ICN(LINE2)
LOOP
FOR EACH HOLE1A DO
DESTROY A SHAPE3 CALLED H1A.ICN(HOLE1A)
LOOP
FOR EACH HOLE2A DO
DESTROY A SHAPE3 CALLED H2A.ICN(HOLE2A)
LOOP
DESTROY EVERY LINE1
DESTROY EVERY LINE2
DESTROY EVERY HOLE1A
DESTROY EVERY HOLE2A
LET NUM.DONE = NUM.DONE+1
LET N = N-1
LET R3 = R3 - 10.0
L1 = 0
ALWAYS
LOOP
END
* END OF SELECT
mv.sim

PROCESS MV GIVEN ICON, X, Y  " GENERAL PURPOSE PROCESS USED FOR ANIMATION
DEFINE ICON AS A POINTER VARIABLE
DEFINE X, Y AS REAL VARIABLES
DEFINE DX, DY AS INTEGER VARIABLES
LET DX = X - LOCATION.X(ICON)
LET DY = Y - LOCATION.Y(ICON)
IF DX NE 0 OR DY NE 0
   LET DISTANCE = SQRT.F(DX*DX + DY*DY)
   LET SPEED = DISTANCE /.TIME
   LET VELOCITY.A(ICON) = VELOCITY.F(SPEED, ARCTAN.F(DY, DX))
   WORK .TIME .SECONDS
   LET VELOCITY.A(ICON) = 0
ALWAYS
END  " END OF MV

mvr.sim

ROUTINE MVR GIVEN ICON, X, Y  " GENERAL PURPOSE ROUTINE USED IN ANIMATION
DEFINE ICON AS A POINTER VARIABLE
DEFINE X, Y AS REAL VARIABLES
DEFINE DX, DY AS INTEGER VARIABLES
LET DX = X - LOCATION.X(ICON)
LET DY = Y - LOCATION.Y(ICON)
IF DX NE 0 OR DY NE 0
   LET DISTANCE = SQRT.F(DX*DX + DY*DY)
   LET SPEED = DISTANCE /.TIME
   LET VELOCITY.A(ICON) = VELOCITY.F(SPEED, ARCTAN.F(DY, DX))
   WORK .TIME .SECONDS
   LET VELOCITY.A(ICON) = 0
ALWAYS
RETURN
END  " END OF MVR

mark.sim

ROUTINE MARKING  " PERFORMS MARKING BEFORE DRILLING
DEFINE X.INITIAL, Y.INITIAL AS REAL VARIABLES
DEFINE X, X1 AND Y AS REAL VARIABLES
IF V1.MAX GT V2.MAX
   LET MAXV = V1.MAX
ELSE
   LET MAXV = V2.MAX
ALWAYS
IF H1.MAX > H2.MAX
   LET MAXH = H1.MAX
ELSE
   LET MAXH = H2.MAX
ALWAYS
LET N.HOLE1A = NUM.EVENTS.1
LET N.HOLE2A = NUM.EVENTS.2
LET N.LINE1 = MAXV
LET N.LINE2 = MAXH
LET X.STEP = (.L/(MAXH+1))
LET Y.STEP = (.H/(MAXV+1))
LET X.INITIAL = 444.0
LET Y.INITIAL = 580.0
LET X1 = .D/(MAXV+1)
CREATE EVERY LINE1
FOR EACH LINE1 DO
   LET U.LINE1(LINE1) = 1
CREATE A SHAPE3 CALLED L1.ICN(LINE1)
LET Y = Y.INITIAL + (Y.STEP)*(LINE1)
LET X = X.INITIAL + X1*(LINE1)
DISPLAY L1.ICN(LINE1) WITH "hline.icn" AT (X,Y)
LOOP
CREATE EVERY LINE2
FOR EACH LINE2 DO
  LET U.LINE2(LINE2) = 1
  CREATE A SHAPE3 CALLED L2.ICN(LINE2)
  LET X = X.INITIAL + X.STEP*(LINE2)
  DISPLAY L2.ICN(LINE2) WITH "vline.icn" AT (X,Y.INITIAL)
LOOP
CREATE EVERY HOLE1A
FOR EACH HOLE1A DO
  LET U.HOLE1A(HOLE1A) = 1
  CREATE A SHAPE3 CALLED H1A.ICN(HOLE1A)
  LET X = X.INITIAL + X.STEP*H1(HOLE1A) + X1*V1(HOLE1A)
  LET Y = Y.INITIAL + (Y.STEP)*V1(HOLE1A)
  DISPLAY H1A.ICN(HOLE1A) WITH "hole1a.icn" AT (X,Y)
  LET X.INITIAL = X
  LET Y.INITIAL = Y
LOOP
LET X.INITIAL = 444.0
LET Y.INITIAL = 580.0
CREATE EVERY HOLE2A
FOR EACH HOLE2A DO
  LET U.HOLE2A(HOLE2A) = 1
  CREATE A SHAPE3 CALLED H2A.ICN(HOLE2A)
  LET X = X.INITIAL + (X.STEP)*H2(HOLE2A) + X1*V2(HOLE2A)
  LET Y = Y.INITIAL + (Y.STEP)*V2(HOLE2A)
  DISPLAY H2A.ICN(HOLE2A) WITH "hole2a.icn" AT (X,Y)
  LET X.INITIAL = X
  LET Y.INITIAL = Y
LOOP
END " END OF MARKING

conveyor.sim

PROCESS CONVEYOR " CONVEYOR BELT ANIMATION
UNTIL SI = 0 DO
  FOR EACH DOT DO
    LET U.DOT(DOT) = 1
    ADD (PI.C/180) *.DEG TO ORIENTATION.A(D.ICN(DOT))
    DISPLAY D.ICN(DOT)
  LOOP
  WAIT .W .SECONDS
LOOP
END " END OF CONVEYOR

confirm1.sim

ROUTINE CONFIRM1 " CONFIRMS WHEN A HOLE HAS BEEN DRILLED
DEFINE XH1 AND YH1 AS REAL VARIABLES
LET XH1 = LOCATION.X(H1A.ICN(K))
LET YH1 = LOCATION.Y(H1A.ICN(K))
CREATE A HOLE CALLED H1B.ICN(K)
DISPLAY H1B.ICN(K) WITH "hole1b.icn" AT (XH1, YH1)
END " END OF CONFIRM1

confirm2.sim

ROUTINE CONFIRM2 " CONFIRMS WHEN HOLE HAS BEEN DRILLED
DEFINE XH1 AND YH1 AS REAL VARIABLES
LET XH1 = LOCATION.X(H2A.ICN(K))
LET YH1 = LOCATION.Y(H2A.ICN(K))
CREATE A HOLE CALLED H2B.ICN(K)
DISPLAY H2B.ICN(K) WITH "hole2b.icn" AT (XH1, YH1)

END  " END OF CONFIRM2

update1.sim

PROCESS UPDATE1 " USED FOR ANIMATION
DEFINE A AS AN INTEGER VARIABLE
DEFINE LX AND LY AS REAL VARIABLES
FOR EACH LINE1 DO
    LET LX = LOCATION.X(L1.ICN(LINE1))-X.STEP*H1(K)
    LET LY = LOCATION.Y(L1.ICN(LINE1)+(Y.STEP*V1(K))/2
    ACTIVATE A MV (L1.ICN(LINE1), LX, LY) NOW
LOOP
FOR EACH LINE2 DO
    LET LX = LOCATION.X(L2.ICN(LINE2))-X.STEP*H1(K)
    LET LY = LOCATION.Y(L2.ICN(LINE2)+(Y.STEP*V1(K))/2
    ACTIVATE A MV (L2.ICN(LINE2), LX, LY) NOW
LOOP
FOR EACH HOLE1A DO
    LET LX = LOCATION.X(H1A.ICN(HOLE1A))-X.STEP*H1(K)
    LET LY = LOCATION.Y(H1A.ICN(HOLE1A)+(Y.STEP*V1(K))/2
    ACTIVATE A MV (H1A.ICN(HOLE1A), LX, LY) NOW
LOOP
FOR EACH HOLE2A DO
    LET LX = LOCATION.X(H2A.ICN(HOLE2A))-X.STEP*H1(K)
    LET LY = LOCATION.Y(H2A.ICN(HOLE2A)+(Y.STEP*V1(K))/2
    ACTIVATE A MV (H2A.ICN(HOLE2A), LX, LY) NOW
LOOP
IF K > 1
    FOR A = 1 TO K-1
        LET LX = LOCATION.X(H1B.ICN(A))-X.STEP*H1(K)
        LET LY = LOCATION.Y(H1B.ICN(A)+(Y.STEP*V1(K))/2
        ACTIVATE A MV (H1B.ICN(A), LX, LY) NOW
    LOOP
ALWAYS
END  " END OF UPDATE1

update2.sim

PROCESS UPDATE2 " USED FOR ANIMATION
DEFINE A AS AN INTEGER VARIABLE
DEFINE LX AND LY AS REAL VARIABLES
FOR EACH LINE1 DO
    LET LX = LOCATION.X(L1.ICN(LINE1))-X.STEP*H2(K)
    LET LY = LOCATION.Y(L1.ICN(LINE1)+(Y.STEP*V2(K))/2
    ACTIVATE A MV (L1.ICN(LINE1), LX, LY) NOW
LOOP
FOR EACH LINE2 DO
    LET LX = LOCATION.X(L2.ICN(LINE2))-X.STEP*H2(K)
    LET LY = LOCATION.Y(L2.ICN(LINE2)+(Y.STEP*V2(K))/2
    ACTIVATE A MV (L2.ICN(LINE2), LX, LY) NOW
LOOP
FOR EACH HOLE1A DO
    LET LX = LOCATION.X(H1A.ICN(HOLE1A))-X.STEP*H2(K)
    LET LY = LOCATION.Y(H1A.ICN(HOLE1A)+(Y.STEP*V2(K))/2
    ACTIVATE A MV (H1A.ICN(HOLE1A), LX, LY) NOW
LOOP
FOR EACH HOLE2A DO
    LET LX = LOCATION.X(H2A.ICN(HOLE2A))-X.STEP*H2(K)
    LET LY = LOCATION.Y(H2A.ICN(HOLE2A)+(Y.STEP*V2(K))/2
    ACTIVATE A MV (H2A.ICN(HOLE2A), LX, LY) NOW
LOOP
FOR EACH HOLE1B DO
LET LX = LOCATION.X(H1B.ICN(HOLE1B))-X.STEP*H2(K)
LET LY = LOCATION.Y(H1B.ICN(HOLE1B))+(Y.STEP*V2(K))/2
ACTIVATE A MV (H1B.ICN(HOLE1B), LX, LY) NOW
LOOP
IF K > 1
    FOR A = 1 TO K-1
        DO
            LET LX = LOCATION.X(H2B.ICN(A))-X.STEP*H2(K)
            LET LY = LOCATION.Y(H2B.ICN(A))+(Y.STEP*V2(K))/2
            ACTIVATE A MV (H2B.ICN(A), LX, LY) NOW
        END
    END
END
"END OF UPDATE2

file1.dat

4
1  1
1  1
3  2
1  -3

file2.dat

2
4  2
-3  2
APPENDIX D

D.1 Code for EDS Event-Based Controller

preamble.sim

PREAMBLE
PROCESSES INCLUDE CCM1, CCM2, CCM3, CCM4, CMMA, INITIATE
DEFINE M1, M2, M3, M4, MA, ERROR AS POINTER VARIABLES
DEFINE SIG1, SIG2, SIG3, SIG4, SIGA AS INTEGER VARIABLES
DEFINE T11, T12, T21, T22, T31, T32, T41, T42, TA1, TA2 AS REAL VARIABLES
DEFINE X1, X2, X3, X4, XA, Y1, Y2, Y3, Y4, YA, I, N, NUM.Plate, NUM.Done AS INTEGER VARIABLES
GRAPHIC ENTITIES INCLUDE STATE AND MESSAGE
* NORMAL OPERATING TIME OF MANIPULATION ODULES
DEFINE .MM1.S TO MEAN 1.0
DEFINE .MM2.S TO MEAN 2.0
DEFINE .MM3A.S TO MEAN 2.0
DEFINE .MM3B.S TO MEAN 0.25
DEFINE .MM4.S TO MEAN 0.25
DEFINE .MM5.S TO MEAN 0.25
DEFINE .MM6.S TO MEAN 0.25
DEFINE .MM7.S TO MEAN 1.0
DEFINE .MM8.S TO MEAN 1.0
DEFINE .MM9.S TO MEAN 1.0
DEFINE .MM10.S TO MEAN 1.0
DEFINE .MMA.S TO MEAN 2.0
DEFINE .CM1.S2 TO MEAN 2.0
DEFINE .CM1.S1 TO MEAN 5.0
DEFINE .CM2.S1 TO MEAN 2.0
DEFINE .CM2.S2 TO MEAN 6.0
DEFINE .CM2.S3 TO MEAN 2.5
DEFINE .CM21.S TO MEAN 0.5
DEFINE .CM3.S1 TO MEAN 2.0
DEFINE .CM3.S2 TO MEAN 4.0
DEFINE .CM4.S TO MEAN 2.0
DEFINE .L TO MEAN 12.125
DEFINE .SEC TO MEAN UNITS
DEFINE .H TO MEAN 50.0
DEFINE MAX.STEP1 TO MEAN 8
DEFINE MAX.STEP2 TO MEAN 5
DEFINE NUM.EVENT.1 TO MEAN 4
DEFINE NUM.EVENT.2 TO MEAN 2

END  "END OF PREAMBLE"

main.sim

MAIN
DEFINE FORM AS A POINTER VARIABLE
DEFINE REPORT AS A POINTER VARIABLE
LET SIG1 = 0
LET SIG2 = 0
LET SIG3 = 0
LET SIG4 = 0
LET SIGA = 0
LET X1 = 20
LET X2 = 20
LET X3 = 20
LET X4 = 20
LET XA = 20
LET Y1 = 750
LET Y2 = 650
LET Y3 = 550
LET Y4 = 450
LET YA = 350

151
LET NUM.DONE = 0
SHOW FORM WITH "data.frm"
IF ACCEPT.F(FORM, 0) EQ "OK"
LET NUM.PLATE = DDVAL.A(DFIELD.F("NUMBER", FORM))
LET VXFORM.V = 1
LET TIMESCALE.V = 100
CALL SETWORLD.R(0.0, 1000.0, 0.0, 1000.0)
ACTIVATE A SELECT NOW
CREATE A MESSAGE CALLED REPORT
DISPLAY REPORT WITH "legend.icn" AT (50, 275)
START SIMULATION
CREATE A MESSAGE CALLED REPORT
DISPLAY REPORT WITH "noerror.icn" AT (550, 275)
READ AS /
ALWAYS

END           " END OF MAIN

select.sim

PROCESS SELECT
DEFINE LABEL AS A POINTER VARIABLE
CREATE A MESSAGE CALLED LABEL
DISPLAY LABEL WITH "title.icn" AT (501, 900)
CREATE A MESSAGE CALLED LABEL
DISPLAY LABEL WITH "namel.icn" AT (X1-10, Y1+10+.H)
CREATE A MESSAGE CALLED LABEL
DISPLAY LABEL WITH "name2.icn" AT (X2-10, Y2+10)
CREATE A MESSAGE CALLED LABEL
DISPLAY LABEL WITH "name3.icn" AT (X3-10, Y3+10)
CREATE A MESSAGE CALLED LABEL
DISPLAY LABEL WITH "name4.icn" AT (X4-10, Y4+10)
CREATE A MESSAGE CALLED LABEL
DISPLAY LABEL WITH "namea.icn" AT (XA-10, YA+10)
FOR J = 1 TO NUM.PLATE+1
DO
ACTIVATE A CCM1 NOW
ACTIVATE A CCM2 NOW
ACTIVATE A CCM3 NOW
ACTIVATE A CCM4 NOW
ACTIVATE A CMMA NOW
WAIT 20 .SEC " A COMPLETE CYCLE TIME
LET NUM.DONE = NUM.DONE+1
LOOP
END

ccm1.sim

PROCESS CCM1
DEFINE A AS A REAL VARIABLE
IF NUM.DONE LT NUM.PLATE
LET T11 = TIME.V
IF NUM.DONE >= 1
CREATE A STATE CALLED M1
DISPLAY M1 WITH "w2u.icn" AT (X1, Y1)
ALWAYS
CREATE A STATE CALLED M1
DISPLAY M1 WITH "yu.icn" AT (X1, Y1)
CREATE A STATE CALLED M1
DISPLAY M1 WITH "s5.icn" AT (X1, Y1+.H)
CREATE A STATE CALLED M1
DISPLAY M1 WITH "wld.icn" AT (X1+5*.L, Y1+.H)
CREATE A STATE CALLED M1
DISPLAY M1 WITH "sd.icn" AT (X1+5*.L, Y1+.H)
WORK A.SEC
LET T12 = TIME.V
IF (T12-T11) <= (A+A/10) AND (T12-T11) >= (A-A/10) " TIME WINDOW
LET SIGA = 1
ELSE
IF(T12-T11) < (A-A/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "errorla.icn" AT (550,275)
READ AS/
STOP
ALWAYS
IF(T12-T11) > (A+A/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "errorlb.icn" AT (550,275)
READ AS/
STOP
ALWAYS
LET XI = LOCATION.X(M1)
LET Y1 = LOCATION.Y(M1)
CREATE A STATE CALLED M1
DISPLAY M1 WITH "wlu.icn" AT (X1+2*.L, Y1-.H)
CREATE A STATE CALLED M1
DISPLAY M1 WITH "s2.icn" AT (XI, Y1-.H)
CREATE A STATE CALLED M1
DISPLAY M1 WITH "su.icn" AT (X1+2*.L, Y1-.H)
LET T11 = TIME.V
UNTIL SIG1 = 1
DO
WAIT 0.25 .SEC
LOOP
LET A = .MMA.S
LET T12 = TIME.V
IF (T12-T11) <= (A+A/10)+0.25 AND (T12-T11) >= (A-A/10)+0.25 " TIME WINDOW
LET SIG1 = 0
ELSE
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "errorcl.icn" AT (550,275)
READ AS/
STOP
ALWAYS
LET T11 = TIME.V
LET XI = LOCATION.X(M1)
LET Y1 = LOCATION.Y(M1)
CREATE A STATE CALLED M1
LET A = .MM2.S
DISPLAY M1 WITH "wld.icn" AT (X1+2*.L, Y1+.H)
CREATE A STATE CALLED M1
DISPLAY M1 WITH "s2.icn" AT (XI, Y1+.H)
CREATE A STATE CALLED M1
DISPLAY M1 WITH "sd.icn" AT (X1+2*.L, Y1+.H)
WORK A .SEC
LET T12 = TIME.V
IF(T12-T11) < (A-A/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "errorla.icn" AT (550,275)
READ AS/
STOP
ALWAYS
IF(T12-T11) > (A+A/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "errorlb.icn" AT (550,275)
READ AS/
STOP
ALWAYS
LET T11 = TIME.V
LET XI = LOCATION.X(M1)
LET Y1 = LOCATION.Y(M1)
CREATE A STATE CALLED M1
DISPLAY M1 WITH "s10.icn" AT (X1, Y1-.H)
WAIT A SEC
LET T12 = TIME.V
IF((T12-T11) < (A-A/10) OR (T12-T11) > (A+A/10))
    CREATE A MESSAGE CALLED ERROR
    DISPLAY ERROR WITH "errorcl.icn" AT (550,275)
    READ AS/
    STOP
ALWAYS
LET X1 = LOCATION.X(M1)+10* .L
LET Y1 = LOCATION.Y(M1)
ALWAYS
IF NUM.DONE >= NUM.PLATE
    LET T11 = TIME.V
    CREATE A STATE CALLED M1
    DISPLAY M1 WITH "s8.icn" AT (X1, Y1)
    WAIT A SEC
    LET T12 = TIME.V
    IF ((T12-T11)GT (A+A/10))
        CREATE A MESSAGE CALLED ERROR
        DISPLAY ERROR WITH "errorcl.icn" AT (550,275)
        READ AS/
        STOP
    ALWAYS
    IF ((T12-T11)LT (A-A/10))
        CREATE A MESSAGE CALLED ERROR
        DISPLAY ERROR WITH "errorcl.icn" AT (550,275)
        write as ' before'
        READ AS/
        STOP
    ALWAYS
ALWAYS
END  " END OF CCM1

ccm2.sim

PROCESS CCM2
DEFINE A AS A REAL VARIABLE
IF NUM.Done < 1
    LET T21 = TIME.V
    LET A = .CM1.S1+.MMA.S
    CREATE A STATE CALLED M2
    DISPLAY M2 WITH "w2u.icn" AT (X2+7*.L, Y2)
    CREATE A STATE CALLED M2
    DISPLAY M2 WITH "s7.icn" AT (X2, Y2)
    CREATE A STATE CALLED M2
    DISPLAY M2 WITH "su.icn" AT (X2+7*.L, Y2)
    LET T21 = TIME.V
    UNTIL SIG2 = 1
    DO
        WAIT 0.25 .SEC
        LOOP
    LET T22 = TIME.V
    IF (T22-T21) <= (A-A/10)+0.25 AND (T22-T21) >= (A-A/10)-0.25 " TIME WINDOW
        LET SIG2 = 0
    LET T21 = TIME.V
    LET X2 = LOCATION.X(M2)
    LET Y2 = LOCATION.Y(M2)
    LET A = .MM3A.S+MM3B.S*MAX.STEP1+.CM21.S*NUM.EVENT.1
    CREATE A STATE CALLED M2
    DISPLAY M2 WITH "wld.icn" AT (X2+6*.L, Y2+.H)
    CREATE A STATE CALLED M2
    DISPLAY M2 WITH "s6.icn" AT (X2, Y2+.H)
    CREATE A STATE CALLED M2
    DISPLAY M2 WITH "sd.icn" AT (X2+6*.L, Y2+.H)
WORK A .SEC
LET T22 = TIME.V
ELSE
  CREATE A MESSAGE CALLED ERROR
  DISPLAY ERROR WITH "error2.icn" AT (550,275)
  READ AS /
  STOP
END
ALWAYS
IF (T22-T21) <= (A+A/10) AND (T22-T21) >= (A-A/10) " TIME WINDOW
  LET SIG4 = 1
ELSE
  IF (T22-T21) < (A-A/10)
    CREATE A MESSAGE CALLED ERROR
    DISPLAY ERROR WITH "error2a.icn" AT (550,275)
    READ AS /
    STOP
  END
  IF (T22-T21) > (A+A/10)
    CREATE A MESSAGE CALLED ERROR
    DISPLAY ERROR WITH "error2b.icn" AT (550,275)
    READ AS /
    STOP
  END
END
ALWAYS
ALWAYS
LET X2 = LOCATION.X(M2)
LET Y2 = LOCATION.Y(M2)
LET A = .CM4.S
CREATE A STATE CALLED M2
DISPLAY M2 WITH "wlu.icn" AT (X2+2*.L, Y2-.H)
CREATE A STATE CALLED M2
DISPLAY M2 WITH "s2.icn" AT (X2, Y2-.H)
CREATE A STATE CALLED M2
DISPLAY M2 WITH "su.icn" AT (X2+2*.L, Y2-.H)
LET T21 = TIME.V
UNTIL SIG2 = 1
DO
  WAIT 0.25 .SEC
  LOOP
LET T22 = TIME.V
IF (T22-T21) <= (A+A/10)+0.25 AND (T22-T21) >= (A-A/10)+0.25
  LET SIG2 = 0
  LET T21 = TIME.V
  LET X2 = LOCATION.X(M2)
  LET Y2 = LOCATION.Y(M2)
  LET A = MM5B.S*MAX.STEP2+CM21.S*NUM.EVENT.2
  CREATE A STATE CALLED M2
  DISPLAY M2 WITH "wld.icn" AT (X2+2*.L, Y2+.H)
  CREATE A STATE CALLED M2
  DISPLAY M2 WITH "s2.icn" AT (X2, Y2+.H)
  CREATE A STATE CALLED M2
  DISPLAY M2 WITH "sd.icn" AT (X2+2*.L, Y2+.H)
  WORK A .SEC
  LET T22 = TIME.V
ELSE
  CREATE A MESSAGE CALLED ERROR
  DISPLAY ERROR WITH "error2.icn" AT (550,275)
  READ AS /
  STOP
END
ALWAYS
IF (T22-T21) <= (A+A/10) AND (T22-T21) >= (A-A/10) " TIME WINDOW
  LET SIG4 = 1
ELSE
  IF (T22-T21) < (A-A/10)
    CREATE A MESSAGE CALLED ERROR
    DISPLAY ERROR WITH "error2a.icn" AT (550,275)
    READ AS /
    STOP
  END
  IF (T22-T21) > (A+A/10)
    CREATE A MESSAGE CALLED ERROR
    DISPLAY ERROR WITH "error2b.icn" AT (550,275)
    READ AS /
    STOP
  END
END
ALWAYS
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "error2b.icn" AT (550, 275)
READ AS /
STOP

ALWAYS
LET T21 = TIME. V
LET X2 = LOCATION.X(M2)
LET Y2 = LOCATION.Y(M2)
LET A = .CM4.S
CREATE A STATE CALLED M2
DISPLAY M2 WITH "s2.icn" AT (X2, Y2-.H)
WAIT A .SEC
LET T22 = TIME. V
IF (T22-T21) >= (A+A/10) OR (T22-T21) <= (A-A/10)
    CREATE A MESSAGE CALLED ERROR
    DISPLAY ERROR WITH "errorc2.icn" AT (550, 275)
    READ AS /
    STOP
ALWAYS
LET X2 = LOCATION.X(M2)+2*.L
LET Y2 = LOCATION.Y(M2)
WAIT A .SEC
LET T22 = TIME. V
IF (T22-T21) >= (A+A/10) OR (T22-T21) <= (A-A/10)
    CREATE A MESSAGE CALLED ERROR
    DISPLAY ERROR WITH "errorc2.icn" AT (550, 275)
    READ AS /
    STOP
ALWAYS
LET A = .MM1.S*2
LET T21 = TIME. V
CREATE A STATE CALLED M2
DISPLAY M2 WITH "s2.icn" AT (X2, Y2)
CREATE A STATE CALLED M2
DISPLAY M2 WITH "wlu.icn" AT (X2+2*.L, Y2)
CREATE A STATE CALLED M2
DISPLAY M2 WITH "su.icn" AT (X2+2*.L, Y2)
WAIT A .SEC
LET T22 = TIME. V
IF (T22-T21) >= (A+A/10) OR (T22-T21) <= (A-A/10)
    CREATE A MESSAGE CALLED ERROR
    DISPLAY ERROR WITH "errorc2.icn" AT (550, 275)
    READ AS /
    STOP
ALWAYS
LET T21 = TIME. V
LET A = .MM3A.S
LET X2 = LOCATION.X(M2)
LET Y2 = LOCATION.Y(M2)
CREATE A STATE CALLED M2
DISPLAY M2 WITH "wld.icn" AT (X2+2*.L, Y2+.H)
CREATE A STATE CALLED M2
DISPLAY M2 WITH "s2.icn" AT (X2, Y2+.H)
CREATE A STATE CALLED M2
DISPLAY M2 WITH "sd.icn" AT (X2+2*.L, Y2+.H)
WORK A .SEC
LET T22 = TIME. V
IF (T22-T21) < (A-A/10)
    CREATE A MESSAGE CALLED ERROR
    DISPLAY ERROR WITH "error2a.icn" AT (550, 275)
    READ AS /
    STOP
ALWAYS
LET SIG2 = 0
LET X2 = LOCATION.X(M2)
LET Y2 = LOCATION.Y(M2)
CREATE A STATE CALLED M2
DISPLAY M2 WITH "wlu.icn" AT (X2+3*.L, Y2-.H)
CREATE A STATE CALLED M2
DISPLAY M2 WITH "s3.icn" AT (X2, Y2-.H)
CREATE A STATE CALLED M2
DISPLAY M2 WITH "su.icn" AT (X2+3*.L, Y2-.H)
LET T21 = TIME.V
UNTIL SIG2 = 1
DO
  WAIT 0.25 .SEC
LOOP
LET T22 = TIME.V
LET A = .MM1.S+MMA.S
IF (T22-T21) <= (A+A/10)+0.25 AND (T22-T21) >= (A-A/10)+0.25
  LET SIG2 = 0
  LET X2 = LOCATION.X(M2)
  LET Y2 = LOCATION.Y(M2)
  LET A = .MM3A.S+MM3B.S*MAX.STEP1+.CM21.S*NUM.EVENT.1
  CREATE A STATE CALLED M2
  DISPLAY M2 WITH "w1d.icn" AT (X2+6*.L, Y2+.H)
  CREATE A STATE CALLED M2
  DISPLAY M2 WITH "s6.icn" AT (X2, Y2+.H)
  CREATE A STATE CALLED M2
  DISPLAY M2 WITH "sd.icn" AT (X2+6*.L, Y2+.H)
  WORK A+1 .SEC
  LET T22 = TIME.V
ELSE
  CREATE A MESSAGE CALLED ERROR
  DISPLAY ERROR WITH "errorc2.icn" AT (550, 275)
  READ AS/
  STOP
END
ALWAYS
IF (T22-T21) <= (A+A/10) AND (T22-T21) >= (A-A/10) " TIME WINDOW
  LET SIG4 = 1
ELSE
  IF(T22-T21) < (A-A/10)
    CREATE A MESSAGE CALLED ERROR
    DISPLAY ERROR WITH "error2a.icn" AT (550, 275)
    READ AS/
    STOP
  ALWAYS
  IF(T22-T21) > (A+A/10)
    CREATE A MESSAGE CALLED ERROR
    DISPLAY ERROR WITH "error2b.icn" AT (550, 275)
    READ AS/
    STOP
  ALWAYS
  LET X2 = LOCATION.X(M2)
  LET Y2 = LOCATION.Y(M2)
  LET A = .CM4.S
  CREATE A STATE CALLED M2
  DISPLAY M2 WITH "w1u.icn" AT (X2+2*.L, Y2-.H)
  CREATE A STATE CALLED M2
  DISPLAY M2 WITH "s2.icn" AT (X2, Y2-.H)
  CREATE A STATE CALLED M2
  DISPLAY M2 WITH "su.icn" AT (X2+2*.L, Y2-.H)
  LET T21 = TIME.V
  UNTIL SIG2 = 1
  DO
    WAIT 0.25 .SEC
  LOOP
  LET T22 = TIME.V
  IF (T22-T21) <= (A+A/10)+0.25 AND (T22-T21) >= (A-A/10)+0.25
    LET SIG2 = 0
    LET X2 = LOCATION.X(M2)
    LET Y2 = LOCATION.Y(M2)
    LET A = .MM3B.S*MAX.STEP2+.CM21.S*NUM.EVENT.2
    CREATE A STATE CALLED M2
DISPLAY M2 WITH "w2d.icn" AT (X2+2*.L, Y2+.H)
CREATE A STATE CALLED M2
DISPLAY M2 WITH "s2.icn" AT (X2, Y2+.H)
CREATE A STATE CALLED M2
DISPLAY M2 WITH "sd.icn" AT (X2+2*.L, Y2+.H)
WORK A .SEC
LET T22 = TIME.V
ELSE
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "errorc2.icn" AT (550,275)
READ AS/
STOP
ALWAYS
IF (T22-T21) <= (A+A/10) AND (T22-T21) >= (A-A/10)
LET SIG4 = 1
ELSE
IF(T22-T21) < (A-A/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "error2a.icn" AT (550,275)
READ AS/
STOP
ALWAYS
IF(T22-T21) > (A+A/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "error2b.icn" AT (550,275)
READ AS/
STOP
ALWAYS
ALWAYS
LET X2 = LOCATION.X(M2)
LET Y2 = LOCATION.Y(M2)
LET A = .CM4.S
CREATE A STATE CALLED M2
DISPLAY M2 WITH "s2.icn" AT (X2, Y2-.H)
LET T21 = TIME.V
UNTIL SIG2 = 1
DO
WAIT 0.25 .SEC
LOOP
LET T22 = TIME.V
IF (T22-T21) <= (A+A/10)+0.25 AND (T22-T21) >= (A-A/10)+0.25
LET SIG2 = 0
ELSE
IF (T22-T21) >= (A+A/10)+0.25
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "errorc2.icn" AT (550,275)
READ AS/
STOP
ALWAYS
IF (T22-T21) <= (A-A/10)+0.25
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "errorc2.icn" AT (550,275)
READ AS/
STOP
ALWAYS
ALWAYS
LET X2 = LOCATION.X(M2)+2*.L
LET Y2 = LOCATION.Y(M2)
ALWAYS
IF NUM.DONE >= NUM.PLATE
LET T21 = TIME.V
LET A = .MM3A.S
CREATE A STATE CALLED M2
DISPLAY M2 WITH "wlu.icn" AT (X2, Y2)
CREATE A STATE CALLED M2
DISPLAY M2 WITH "su.icn" AT (X2, Y2)
CREATE A STATE CALLED M2
DISPLAY M2 WITH "s2.icn" AT (X2, Y2+.H)
Appendix D

WORK A.SEC
LET T22 = TIME.V
IF (T22-T21) <= (A+A/10) AND (T22-T21) >= (A-A/10) " TIME WINDOW
LET SIGA = 1
ELSE
IF(T22-T21) < (A-A/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "error2a.icn" AT (550,275)
READ AS/
STOP
ALWAYS
IF(T22-T21) > (A+A/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "error2b.icn" AT (550,275)
READ AS/
STOP
ALWAYS
ALWAYS
LET T21 = TIME.V
LET X2 = LOCATION.X(M2)
LET Y2 = LOCATION.Y(M2)
LET A = .MMA.S+.CM3.S2
CREATE A STATE CALLED M2
DISPLAY M2 WITH "wld.icn" AT (X2+2*.L, Y2)
CREATE A STATE CALLED M2
DISPLAY M2 WITH "s6.icn" AT (X2+2*L, Y2-.H)
CREATE A STATE CALLED M2
DISPLAY M2 WITH "sd.icn" AT (X2+2*.L, Y2)
WAIT A .SEC
LET T22 = TIME.V
IF (T22-T21) GT (A+A/10) OR (T22-T21) LT (A-A/10)
DISPLAY ERROR WITH "errorc2.icn" AT (550,275)
READ AS/
STOP
ALWAYS
ALWAYS
END " END OF COM2

ccm3.sim

PROCESS CCM3
DEFINE A AS A REAL VARIABLE
IF NUM.DONE LT 1
LET T31 = TIME.V
CREATE A STATE CALLED M3
DISPLAY M3 WITH "sl9.icn" AT (X3, Y3)
WAIT A .SEC
LET T32 = TIME.V
IF (T32-T31) >= (A+A/10) OR (T32-T31) <= (A-A/10) " TIME WINDOW
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "errorc3.icn" AT (550, 275)
READ AS/
STOP
ALWAYS
LET X3 = LOCATION.X(M3) + A*.L
LET Y3 = LOCATION.Y(M3)
ALWAYS
IF NUM.DONE >= 1 AND NUM.DONE < NUM.PLATE
LET T31 = TIME.V
LET A = .MM1.S*2
CREATE A STATE CALLED M3
DISPLAY M3 WITH "wlu.icn" AT (X3+A*.L, Y3)
CREATE A STATE CALLED M3
DISPLAY M3 WITH "s2.icn" AT (X3, Y3)
CREATE A STATE CALLED M3
DISPLAY M3 WITH "su.icn" AT (X3+A*.L, Y3)
WAIT A .SEC
LET T32 = TIME.V
IF(T32-T31) < (A-A/10) OR (T32-T31) > (A+A/10)
    CREATE A MESSAGE CALLED ERROR
    DISPLAY ERROR WITH "error3.icn" AT (550,275)
    READ AS /
STOP
ALWAYS
LET T31 = TIME.V
LET X3 = LOCATION.X(M3)
LET Y3 = LOCATION.Y(M3)
LET A = .MM7.S
CREATE A STATE CALLED M3
DISPLAY M3 WITH "w1d.icn" AT (X3+A*.L, Y3+.H)
CREATE A STATE CALLED M3
DISPLAY M3 WITH "s2.icn" AT (X3, Y3+.H)
CREATE A STATE CALLED M3
DISPLAY M3 WITH "sd.icn" AT (X3+A*.L, Y3+.H)
WORK A .SEC
LET T32 = TIME.V
IF (T32-T31) <= (A+A/10) AND (T32-T31) >= (A-A/10)
    LET SIG3 = 0
ELSE
    IF(T32-T31) < (A-A/10)
        CREATE A MESSAGE CALLED ERROR
        DISPLAY ERROR WITH "error3a.icn" AT (550,275)
        READ AS /
        STOP
ALWAYS
IF(T32-T31) > (A+A/10)
    CREATE A MESSAGE CALLED ERROR
    DISPLAY ERROR WITH "error3b.icn" AT (550, 275)
    READ AS /
    STOP
ALWAYS
ALWAYS
LET T31 = TIME.V
LET X3 = LOCATION.X(M3)
LET Y3 = LOCATION.Y(M3)
LET A = .MMA.S+.MM1.S
CREATE A STATE CALLED M3
DISPLAY M3 WITH "w2u.icn" AT (X3+A*.L, Y3-.H)
CREATE A STATE CALLED M3
DISPLAY M3 WITH "s3.icn" AT (X3, Y3-.H)
CREATE A STATE CALLED M3
DISPLAY M3 WITH "su.icn" AT (X3+A*.L, Y3-.H)
UNTIL SIG3 = 1
DO
    WAIT 0.25 .SEC
LOOP
LET T32 = TIME.V
IF (T32-T31) <= (A+A/10)+0.25 AND (T32-T31) >= (A-A/10)+0.25
    LET T31 = TIME.V
    LET X3 = LOCATION.X(M3)
    LET Y3 = LOCATION.Y(M3)
    LET A = .MM7.S+.MM8.S*2
    CREATE A STATE CALLED M3
    DISPLAY M3 WITH "w2d.icn" AT (X3+A*.L, Y3+.H)
    CREATE A STATE CALLED M3
    DISPLAY M3 WITH "s4.icn" AT (X3, Y3+.H)
    CREATE A STATE CALLED M3
    DISPLAY M3 WITH "sd.icn" AT (X3+A*.L, Y3+.H)
    WORK A .SEC
    LET T32 = TIME.V
ELSE
    CREATE A MESSAGE CALLED ERROR
    DISPLAY ERROR WITH "errorc3.icn" AT (550,275)
    READ AS /
    STOP
Appendix D

161

ALWAYS
IF(T32-T31) < (A-A/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "error3a.icn" AT (550,275)
READ AS/
STOP
ALWAYS
IF(T32-T31) > (A+A/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "error3b.icn" AT (550,275)
READ AS/
STOP
ALWAYS
LET T31 = TIME.V
LET X3 = LOCATION.X(M3)
LET Y3 = LOCATION.Y(M3)
CREATE A STATE CALLED M3
DISPLAY M3 WITH "s8.icn" AT (X3, Y3-.H)
WORK A.SEC
LET T32 = TIME.V
IF(T32-T31) < (A-A/10) OR (T32-T31) > (A+A/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "errorc3.icn" AT (550,275)
READ AS/
STOP
ALWAYS
LET X3 = LOCATION.X(M3)+A*L
LET Y3 = LOCATION.Y(M3)
ALWAYS
IF NUMDONE >= NUMPLATE
LET T31 = TIME.V
LET A = .MM7.S
CREATE A STATE CALLED M3
DISPLAY M3 WITH "wlu.icn" AT (X3, Y3)
CREATE A STATE CALLED M3
DISPLAY M3 WITH "su.icn" AT (X3, Y3)
CREATE A STATE CALLED M3
DISPLAY M3 WITH "s2.icn" AT (X3, Y3-.H)
CREATE A STATE CALLED M3
DISPLAY M3 WITH "wldicn" AT (X3+A*L, Y3+.H)
CREATE A STATE CALLED M3
DISPLAY M3 WITH "sd.icn" AT (X3+A*L, Y3+.H)
WORK A.SEC
LET T32 = TIME.V
IF (T32-T31) <= (A+A/10) AND (T32-T31) >= (A-A/10)
LET SIG3 = 0
ELSE
IF(T32-T31) < (A-A/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "error3a.icn" AT (550,275)
READ AS/
STOP
ALWAYS
IF(T32-T31) > (A+A/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "error3b.icn" AT (550,275)
READ AS/
STOP
ALWAYS
ALWAYS
LET T31 = TIME.V
LET X3 = LOCATION.X(M3)
LET Y3 = LOCATION.Y(M3)
LET A = .MMA.S
CREATE A STATE CALLED M3
DISPLAY M3 WITH "wlu.icn" AT (X3+A*L, Y3-.H)
CREATE A STATE CALLED M3
DISPLAY M3 WITH "s2.icn" AT (X3, Y3-.H)
CREATE A STATE CALLED M3
DISPLAY M3 WITH "su.icn" AT (X3+A*.L, Y3-.H)
UNTIL SIG3 = 1
DO
  WAIT 0.25 .SEC
LOOP
LET T32 = TIME.V
IF (T32-T31) <= (A+A/10)+0.25 AND (T32-T31) >= (A-A/10)+0.25
LET X3 = LOCATION.X(M3)
LET Y3 = LOCATION.Y(M3)
CREATE A STATE CALLED M3
DISPLAY M3 WITH "w2d.icn" AT (X3+A*.L, Y3+.H)
CREATE A STATE CALLED M3
DISPLAY M3 WITH "s4.icn" AT (X3, Y3+.H)
CREATE A STATE CALLED M3
DISPLAY M3 WITH "sd.icn" AT (X3+A*.L, Y3+.H)
WORK A .SEC
LET T32 = TIME.V
ELSE
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "errorc3.icn" AT (550, 275)
READ AS/
STOP
ALWAYS
IF(T32-T31) < (A-A/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "error3a.icn" AT (550, 275)
READ AS/
STOP
ALWAYS
IF(T32-T31) > (A+A/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "error3b.icn" AT (550, 275)
READ AS/
STOP
ALWAYS
END " END OD CCM3

ccm4.sim

PROCESS CCM4
DEFINE A AS A REAL VARIABLE
IF NUM.DONE LT NUM.PLATE
  LET A = .CM1.S1+.MMA.S+.CM2.S2
CREATE A STATE CALLED M4
DISPLAY M4 WITH "s13.icn" AT (X4, Y4)
CREATE A STATE CALLED M4
DISPLAY M4 WITH "wlu.icn" AT (X4+A*.L, Y4)
CREATE A STATE CALLED M4
DISPLAY M4 WITH "su.icn" AT (X4+A*.L, Y4)
LET T41 = TIME.V
UNTIL SIG4 = 1
DO
  WAIT 0.25 .SEC
LOOP
LET T42 = TIME.V
IF (T42-T41) <= (A+A/10)+0.25 AND (T42-T41) >= (A-A/10)+0.25
LET SIG4 = 0
ELSE
IF(T42-T41) < (A-A/10)+0.25
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "errorc4.icn" AT (550,275)
READ AS/
STOP
" TIME WINDOW
Appendix D

163

ALWAYS
IF(T42-T41) > (A+A/10)+0.25
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "error4.icn" AT (550,275)
READ AS/
STOP

ALWAYS
LET T41 = TIME.V
LET X4 = LOCATION.X(M4)
LET Y4 = LOCATION.Y(M4)
LET A = .MM9.5+.MM10.S
CREATE A STATE CALLED M4
DISPLAY M4 WITH "w1.d.icn" AT (X4+2*L, Y4+.H)
CREATE A STATE CALLED M4
DISPLAY M4 WITH "s2.icn" AT (X4, Y4+.H)
CREATE A STATE CALLED M4
DISPLAY M4 WITH "sd.icn" AT (X4+2*L, Y4+.H)
WORK A_SEC
LET T42 = TIME.V
IF (T42-T41) <= (A+A/10) AND (T42-T41) >= (A-A/10)
LET SIG2 = 1
ELSE
IF(T42-T41) < (A-A/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "error4a.icn" AT (550,275)
READ AS/
STOP

ALWAYS
IF(T42-T41) > (A+A/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "error4b.icn" AT (550,275)
READ AS/
STOP

ALWAYS
LET T41 = TIME.V
LET X4 = LOCATION.X(M4)
LET Y4 = LOCATION.Y(M4)
LET A = .CM2.S3
CREATE A STATE CALLED M4
DISPLAY M4 WITH "wlu.icn" AT (X4+2*L, Y4-.H)
CREATE A STATE CALLED M4
DISPLAY M4 WITH "s2.icn" AT (X4, Y4-.H)
CREATE A STATE CALLED M4
DISPLAY M4 WITH "su.icn" AT (X4+2*L, Y4-.H)
LET T41 = TIME.V
UNTIL SIG4 = 1
DO
WAIT 0.25_SEC
LOOP
LET T42 = TIME.V
IF (T42-T41) <= (A+A/10)+0.25 AND (T42-T41) >= (A-A/10)+0.25
LET SIG4 = 0
ELSE
IF(T42-T41) < (A-A/10)+0.25
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "error4.icn" AT (550,275)
READ AS/
STOP

ALWAYS
IF(T42-T41) > (A+A/10)+0.25
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "error4.icn" AT (550,275)
READ AS/
STOP

ALWAYS
LET T41 = TIME.V
LET X4 = LOCATION.X(M4)
LET Y4 = LOCATION.Y(M4)
CREATE A STATE CALLED M4
DISPLAY M4 WITH "wld.icn" AT (X4+2*.L, Y4+.H)
CREATE A STATE CALLED M4
DISPLAY M4 WITH "s2.icn" AT (X4, Y4+.H)
CREATE A STATE CALLED M4
DISPLAY M4 WITH "sd.icn" AT (X4+2*.L, Y4+.H)
WORK A.SEC
LET T42 = TIME.V
IF(T42-T41) < (A-A/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "error4a.icn" AT (550, 275)
READ AS/
STOP
ALWAYS
IF(T42-T41) > (A+A/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "error4b.icn" AT (550, 275)
READ AS/
STOP
ALWAYS
LET SIG2 = 1
LET X4 = LOCATION.X(M4)
LET Y4 = LOCATION.Y(M4)-.H
ALWAYS
IF NUM.DONE >= NUM.PLATE
LET T41 = TIME.V
CREATE A STATE CALLED M4
DISPLAY M4 WITH "s8.icn" AT (X4, Y4)
WAIT A .SEC
LET T42 = TIME.V
IF (T42-T41) <= (A+A/10) AND (T42-T41) >= (A-A/10) " TIME WINDOW
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "error4c.icn" AT (550, 275)
READ AS/
STOP
ALWAYS
END " END OF CCM4

ccma.sim

PROCESS CMMA
DEFINE A AS A REAL VARIABLE
IF NUM.DONE LT NUM.PLATE
LET A = .CM1.S1
CREATE A STATE CALLED MA
DISPLAY MA WITH "wlu.icn" AT (XA+5*.L, YA)
CREATE A STATE CALLED MA
DISPLAY MA WITH "s5.icn" AT (XA, YA)
CREATE A STATE CALLED MA
DISPLAY MA WITH "su.icn" AT (XA+5*.L, YA)
LET TA1 = TIME.V
UNTIL SIGA = 1
DO
WAIT 0.25 .SEC
LOOP
LET TA2 = TIME.V
IF (TA2-TA1) <= (A+A/10) AND (TA2-TA1) >= (A-A/10) " TIME WINDOW
LET SIGA = 0
LET TA1 = TIME.V
LET XA = LOCATION.X(MA)
LET YA = LOCATION.Y(MA)
LET A = .MMA.S
CREATE A STATE CALLED MA
DISPLAY MA WITH "wld.icn" AT (XA+2*.L, YA+.H)
CREATE A STATE CALLED MA
DISPLAY MA WITH "s2.icn" AT (XA, YA+.H)
CREATE A STATE CALLED MA
DISPLAY MA WITH "sd.icn" AT (XA+2*.L, YA+.H)
WORK A .SEC
LET TA2 = TIME.V
ELSE
   CREATE A MESSAGE CALLED ERROR
   DISPLAY ERROR WITH "errorca.icn" AT (550,275)
   READ AS/
   STOP
END IF
ALWAYS
IF (TA2-TA1) <= (A+A/10) AND (TA2-TA1) >= (A-A/10) * TIME WINDOW
   LET SIG1 = 1
   LET SIG2 = 1
   LET SIG3 = 1
ELSE
IF (TA2-TA1) < (A-A/10)
   CREATE A MESSAGE CALLED ERROR
   DISPLAY ERROR WITH "erroral.icn" AT (550, 275)
   READ AS/
   STOP
END IF
IF (TA2-TA1) > (A+A/10)
   CREATE A MESSAGE CALLED ERROR
   DISPLAY ERROR WITH "error2.icn" AT (550, 275)
   READ AS/
   STOP
END IF
ALWAYS
ALWAYS
LET TA1 = TIME.V
LET XA = LOCATION.X(MA)
LET YA = LOCATION.Y(MA)
LET A = .CM2.S2+.CM4.S*2+.CM2.S3
CREATE A STATE CALLED MA
DISPLAY MA WITH "s12.icn" AT (XA, YA-.H)
WAIT A .SEC
LET TA2 = TIME.V
IF (TA2-TA1) >= (A+A/10) OR (TA2-TA1) <= (A-A/10)
   CREATE A MESSAGE CALLED ERROR
   DISPLAY ERROR WITH "errorca.icn" AT (550,275)
   READ AS/
   STOP
END IF
ALWAYS
LET XA = LOCATION.X(MA)+12*.L
LET YA = LOCATION.Y(MA)
ALWAYS
IF NUM.DONE >= NUM.PLATE
   LET A = .CM2.S1
   CREATE A STATE CALLED MA
   DISPLAY MA WITH "wlu.icn" AT (XA+2*.L, YA)
   CREATE A STATE CALLED MA
   DISPLAY MA WITH "s2.icn" AT (XA, YA)
   CREATE A STATE CALLED MA
   DISPLAY MA WITH "su.icn" AT (XA+2*.L, YA)
   LET TA1 = TIME.V
   UNTIL SIGA = 1
   DO
      WAIT 0.25 .SEC
   LOOP
   LET TA2 = TIME.V
   IF (TA2-TA1) <= (A+A/10) AND (TA2-TA1) >= (A-A/10)
      LET SIGA = 0
      LET TA1 = TIME.V
      LET XA = LOCATION.X(MA)
      LET YA = LOCATION.Y(MA)
      LET A = .MMA.S
      CREATE A STATE CALLED MA
DISPLAY MA WITH "wld.icn" AT (XA+2*L, YA+H)
CREATE A STATE CALLED MA
DISPLAY MA WITH "s2.icn" AT (XA, YA+H)
CREATE A STATE CALLED MA
DISPLAY MA WITH "sd.icn" AT (XA+2*L, YA+H)
WORK A .SEC
LET TA2 = TIME.V
ELSE
   CREATE A MESSAGE CALLED ERROR
   DISPLAY ERROR WITH "errorca.icn" AT (550,275)
   READ AS /
   STOP
ALWAYS
IF (TA2-TA1) <= (A+A/10) AND (TA2-TA1) >= (A-A/10) " TIME WINDOW
   LET SIG3 = 1
ELSE
   IF(TA2-TA1) < (A-A/10)
      CREATE A MESSAGE CALLED ERROR
      DISPLAY ERROR WITH "erroral.icn" AT (550, 275)
      READ AS /
      STOP
   ALWAYS
   IF(TA2-TA1) > (A+A/10)
      CREATE A MESSAGE CALLED ERROR
      DISPLAY ERROR WITH "errora2.icn" AT (550, 275)
      READ AS /
      STOP
   ALWAYS
ALWAYS
LET TA1 = TIME.V
LET XA = LOCATION.X(MA)
LET YA = LOCATION.Y(MA)
LET A = .CM3.S2
CREATE A STATE CALLED MA
DISPLAY MA WITH "s4.icn" AT (XA, YA-H)
WAIT A .SEC
LET TA2 = TIME.V
IF (TA2-TA1) GT (A+A/10) OR (TA2-TA1) LT (A-A/10)
   CREATE A MESSAGE CALLED ERROR
   DISPLAY ERROR WITH "errorca.icn" AT (550, 275)
   READ AS /
   STOP
ALWAYS
ALWAYS
ALWAYS
ALWAYS
ALWAYS
END " END OF CMMA

D.2 Code for Control Signal of CM1

preamble.sim

PREAMBLE
PROCESSES INCLUDE MM1, MM2, CM1
DEFINE S1 AND S2 AS INTEGER VARIABLES
DEFINE ERROR, LABEL, REPORT AS pointer VARIABLES
DEFINE X, Y AS REAL VARIABLES
GRAPHIC ENTITIES INCLUDE STATE AND MESSAGE
DEFINE L TO MEAN 50.0
DEFINE H TO MEAN 50.0
DEFINE .M1.S TO MEAN 1
DEFINE .M1.W TO MEAN 2
DEFINE .M1.WL TO MEAN 11
DEFINE .M2.S TO MEAN 2
DEFINE .M2.W1 TO MEAN 2  
DEFINE .M2.W2 TO MEAN 3  
DEFINE .M2.WL TO MEAN 10  
DEFINE .CYCLE.TIME TO MEAN 19.0  
END  
"END OF PREAMBLE

main.sim

MAIN

LET X = 30  
LET Y = 750  
LET SI = 1  
LET VXFORM.V = 1  
LET TIMESCALE.V = 100  
CALL SETWORLD.R(0.0,1000.0, 0.0, 1000.0)  
CREATE A MESSAGE CALLED REPORT  
DISPLAY REPORT WITH "title.icn" AT (501,900)  
CREATE A MESSAGE CALLED REPORT  
DISPLAY REPORT WITH "heading.icn" AT (501,850)  
ACTIVATE A MM1 NOW  
ACTIVATE A MM2 NOW  
ACTIVATE A CM1 NOW  
START SIMULATION  
CREATE A MESSAGE CALLED REPORT  
DISPLAY REPORT WITH "legend.icn" AT (50,275)  
READ AS/

END  
"END OF MAIN

m1.sim

PROCESS MM1

DEFINE A11 AND A12 AS INTEGER VARIABLES  
DEFINE M1 AS A POINTER VARIABLE  
DEFINE U AS A INTEGER VARIABLE  
CREATE A MESSAGE CALLED LABEL  
DISPLAY LABEL WITH "namel.icn" AT (30, Y+25)  
DEFINE X1, Y1 AS REAL VARIABLES  
LET SI = 1  
LET All = TIME.V  
LET U = . M1.S  
CREATE A STATE CALLED M1  
DISPLAY M1 WITH "sdl.icn" AT (X+U*.L, Y)  
CREATE A STATE CALLED M1  
DISPLAY M1 WITH "sl.icn" AT (X, Y)  
CREATE A STATE CALLED M1  
DISPLAY M1 WITH "s.icn" AT (X+U*.L, Y)  
WORK U UNITS  
LET A12 = TIME.V  
IF (A12-A11) <= (U-U/10) AND (A12-A11) >= (U-U/10) " TIME WINDOW  
LET SI = 1  
LET A11 = TIME.V  
LET U = .M1.S  
LET X1 = LOCATION.X(M1)  
LET Y1 = LOCATION.Y(M1)  
CREATE A STATE CALLED M1  
DISPLAY M1 WITH "wlu.icn" AT (X1+U*.L, Y1-2*.H)  
CREATE A STATE CALLED M1  
DISPLAY M1 WITH "sl.icn" AT (X1, Y1-2*.H)  
CREATE A STATE CALLED M1  
DISPLAY M1 WITH "su.icn" AT (X1+U*.L, Y1-2*.H)  
WORK U UNITS  
LET A12 = TIME.V
ELSE

IF(A12-A11) < (U-U/10)
  CREATE A MESSAGE CALLED ERROR
  DISPLAY ERROR WITH "error1a.icn" AT (50,275)
  READ AS /
  STOP
ALWAYS

IF(A12-A11) > (U+U/10)
  CREATE A MESSAGE CALLED ERROR
  DISPLAY ERROR WITH "error1b.icn" AT (50,275)
  READ AS /
  STOP
ALWAYS

ALWAYS

IF (A12-A11) <= (U+U/10) AND (A12-A11) >= (U-U/10) " TIME WINDOW
  LET S1 = 0
  LET A11 = TIME.V
  LET U = M1.W
  LET X1 = LOCATION.X(M1)
  LET Y1 = LOCATION.Y(M1)
  CREATE A STATE CALLED M1
  DISPLAY M1 WITH "wlu.icn" AT (X1+U*.L, Y1+.H)
  CREATE A STATE CALLED M1
  DISPLAY M1 WITH "s2.icn" AT (X1, Y1+.H)
  CREATE A STATE CALLED M1
  DISPLAY M1 WITH "su.icn" AT (X1+U*.L, Y1+.H)
  WAIT U UNITS
  LET A12 = TIME.V

ELSE

IF(A12-A11) < (U-U/10)
  CREATE A MESSAGE CALLED ERROR
  DISPLAY ERROR WITH "error1a.icn" AT (50,275)
  READ AS /
  STOP
ALWAYS

IF(A12-A11) > (U+U/10)
  CREATE A MESSAGE CALLED ERROR
  DISPLAY ERROR WITH "error1b.icn" AT (50,275)
  READ AS /
  STOP
ALWAYS

ALWAYS

IF (A12-A11) <= (U+U/10) AND (A12-A11) >= (U-U/10) " TIME WINDOW
  LET S1 = 1
  LET A11 = TIME.V
  LET U = M1.S
  LET X1 = LOCATION.X(M1)
  LET Y1 = LOCATION.Y(M1)
  CREATE A STATE CALLED M1
  DISPLAY M1 WITH "wld.icn" AT (X1+U*.L, Y1+.H)
  CREATE A STATE CALLED M1
  DISPLAY M1 WITH "sl.icn" AT (X1, Y1+.H)
  CREATE A STATE CALLED M1
  DISPLAY M1 WITH "sd.icn" AT (X1+U*.L, Y1+.H)
  WORK U UNITS
  LET A12 = TIME.V

ELSE

IF(A12-A11) < (U-U/10) OR (A12-A11) > (U+U/10)
  CREATE A MESSAGE CALLED ERROR
  DISPLAY ERROR WITH "error1c.icn" AT (50,275)
  READ AS /
  STOP
ALWAYS

ALWAYS

IF (A12-A11) <= (U+U/10) AND (A12-A11) >= (U-U/10) " TIME WINDOW
  LET S1 = 0
  LET A11 = TIME.V
  LET U = M1.W
  LET X1 = LOCATION.X(M1)
LET Y1 = LOCATION.Y(M1)
CREATE A STATE CALLED M1
DISPLAY M1 WITH "wld.icn" AT (X1+U*.L, Y1-.H)
CREATE A STATE CALLED M1
DISPLAY M1 WITH "s2.icn" AT (X1, Y1-.H)
CREATE A STATE CALLED M1
DISPLAY M1 WITH "sd.icn" AT (X1+U*.L, Y1-.H)
WAIT U UNITS
LET A12 = TIME.V
ELSE
IF(A12-A11) < (U-U/10)
  CREATE A MESSAGE CALLED ERROR
  DISPLAY ERROR WITH "errorla.icn" AT (50, 275)
  READ AS /
  STOP
ALWAYS
IF(A12-A11) > (U+U/10)
  CREATE A MESSAGE CALLED ERROR
  DISPLAY ERROR WITH "errorlb.icn" AT (50, 275)
  READ AS /
  STOP
ALWAYS
IF (A12-A11) <= (U+U/10) AND (A12-A11) >= (U-U/10) " TIME WINDOW
LET S1 = 1
LET A11 = TIME.V
LET U = M1.S
LET X1 = LOCATION.X(M1)
LET Y1 = LOCATION.Y(M1)
CREATE A STATE CALLED M1
DISPLAY M1 WITH "wlu.icn" AT (X1+U*.L, Y1-.H)
CREATE A STATE CALLED M1
DISPLAY M1 WITH "sl.icn" AT (X1, Y1-.H)
CREATE A STATE CALLED M1
DISPLAY M1 WITH "su.icn" AT (X1+U*.L, Y1-.H)
WORK U UNITS
LET A12 = TIME.V
ELSE
IF(A12-A11) < (U-U/10) OR (A12-A11) > (U+U/10)
  CREATE A MESSAGE CALLED ERROR
  DISPLAY ERROR WITH "errorcl.icn" AT (50, 275)
  READ AS /
  STOP
ALWAYS
IF (A12-A11) <= (U+U/10) AND (A12-A11) >= (U-U/10) " TIME WINDOW
LET S1 = 0
LET A11 = TIME.V
LET U = M1.WL
LET X1 = LOCATION.X(M1)
LET Y1 = LOCATION.Y(M1)
CREATE A STATE CALLED M1
DISPLAY M1 WITH "s11.icn" AT (X1, Y1+.H)
WAIT U UNITS
LET A12 = TIME.V
ELSE
IF(A12-A11) < (U-U/10)
  CREATE A MESSAGE CALLED ERROR
  DISPLAY ERROR WITH "errorla.icn" AT (50, 275)
  READ AS /
  STOP
ALWAYS
IF(A12-A11) > (U+U/10)
  CREATE A MESSAGE CALLED ERROR
  DISPLAY ERROR WITH "errorlb.icn" AT (50, 275)
  READ AS /
  STOP
ALWAYS
ALWAYS
IF (A12-A11) <= (U+U/10) AND (A12-A11) >= (U-U/10) " TIME WINDOW
LET S1 = 0
ELSE
IF (A12-A11) < (U-U/10) OR (A12-A11) > (U+U/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "error1.icn" AT (50, 275)
READ AS /
STOP
ALWAYS
END "END OF M1

m2.sim

PROCESS M2
DEFINE U, A21 AND A22 AS INTEGER VARIABLES
DEFINE M2 AS A POINTER VARIABLE
DEFINE X2, Y2 AS REAL VARIABLES
CREATE A MESSAGE CALLED LABEL
DISPLAY LABEL WITH "name2.icn" AT (30, Y-250+25)
LET S2 = 0
LET A21 = TIME.V
LET U = .M2.W1
CREATE A STATE CALLED M2
DISPLAY M2 WITH "wlu.icn" AT (X+U*.L, Y-250)
CREATE A STATE CALLED M2
DISPLAY M2 WITH "s2.icn" AT (X, Y-250)
CREATE A STATE CALLED M2
DISPLAY M2 WITH "su.icn" AT (X+U*.L, Y-250)
WAIT U UNITS
LET A22 = TIME.V
IF (A22-A21) <= (U+U/10) AND (A22-A21) >= (U-U/10) " TIME WINDOW
LET S2 = 1
LET A21 = TIME.V
LET U = 2
LET X2 = LOCATION.X(M2)
LET Y2 = LOCATION.Y(M2)
CREATE A STATE CALLED M2
DISPLAY M2 WITH "wld.icn" AT (X2+U*.L, Y2+.H)
CREATE A STATE CALLED M2
DISPLAY M2 WITH "s2.icn" AT (X2, Y2+.H)
CREATE A STATE CALLED M2
DISPLAY M2 WITH "sd.icn" AT (X2+U*.L, Y2-.H)
WORK U UNITS
LET A22 = TIME.V
ELSE
IF (A22-A21) < (U-U/10) OR (A22-A21) > (U+U/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "error2.icn" AT (50, 275)
READ AS /
STOP
ALWAYS
IF (A22-A21) <= (U+U/10) AND (A22-A21) >= (U-U/10) " TIME WINDOW
LET S2 = 0
LET A21 = TIME.V
LET U = .M2.W2
LET X2 = LOCATION.X(M2)
LET Y2 = LOCATION.Y(M2)
CREATE A STATE CALLED M2
DISPLAY M2 WITH "wld.icn" AT (X2+U*.L, Y2-.H)
CREATE A STATE CALLED M2
DISPLAY M2 WITH "s3.icn" AT (X2, Y2-.H)
CREATE A STATE CALLED M2
DISPLAY M2 WITH "sd.icn" AT (X2+U*.L, Y2-.H)
WAIT U UNITS
LET A22 = TIME.V
ELSE

IF (A22 - A21) < (U - U/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "error1a.icn" AT (50, 275)
READ AS /
STOP

ALWAYS
IF (A22 - A21) > (U + U/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "error1b.icn" AT (50, 275)
READ AS /
STOP

ALWAYS

ALWAYS
IF (A22 - A21) <= (U + U/10) AND (A22 - A21) >= (U - U/10) " TIME WINDOW
LET S2 = 1
LET A21 = TIME.V
LET U = M2.S
LET X2 = LOCATION.X(M2)
LET Y2 = LOCATION.Y(M2)
CREATE A STATE CALLED M2
DISPLAY M2 WITH "wlu.icn" AT (X2 + U*.L, Y2 - .H)
CREATE A STATE CALLED M2
DISPLAY M2 WITH "s2.icn" AT (X2, Y2 - .H)
CREATE A STATE CALLED M2
DISPLAY M2 WITH "su.icn" AT (X2 + U*.L, Y2 - .H)
WORK U UNITS
LET A22 = TIME.V

ELSE

IF (A22 - A21) < (U - U/10) OR (A22 - A21) > (U + U/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "error2.icn" AT (50, 275)
READ AS /
STOP

ALWAYS

ALWAYS
IF (A22 - A21) <= (U + U/10) AND (A22 - A21) >= (U - U/10) " TIME WINDOW
LET S2 = 0
LET A21 = TIME.V
LET U = M2.WL
LET X2 = LOCATION.X(M2)
LET Y2 = LOCATION.Y(M2)
CREATE A STATE CALLED M2
DISPLAY M2 WITH "s10.icn" AT (X2, Y2 + .H)
WAIT U UNITS
LET A22 = TIME.V

ELSE

IF (A22 - A21) < (U - U/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "error1a.icn" AT (50, 275)
READ AS /
STOP

ALWAYS
IF (A22 - A21) > (U + U/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "error1b.icn" AT (50, 275)
READ AS /
STOP

ALWAYS

ALWAYS
IF (A22 - A21) <= (U + U/10) AND (A22 - A21) >= (U - U/10) " TIME WINDOW
LET S2 = 0
ELSE

IF (A22 - A21) < (U - U/10) OR (A22 - A21) > (U + U/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "error2.icn" AT (50, 275)
READ AS /
STOP

ALWAYS
ALWAYS
END " END OF M2

**cm1.sim**

PROCESS CM1
DEFINE MM AS A POINTER VARIABLE
CREATE A MESSAGE CALLED LABEL
DISPLAY LABEL WITH "name.icn" AT (30, Y-400+25)
DEFINE XM, YM AS REAL VARIABLES
LET XM = X
LET YM = Y-400
LABEL
WHILE TIME.V LT .CYCLE.TTIME
DO
WHILE S1 = 1 OR S2 = 1
DO
WAIT 1 UNIT
CREATE A STATE CALLED MM
DISPLAY MM WITH "s1.icn" AT (XM, YM)
LET XM = LOCATION.X(MM)+.L
LET YM = LOCATION.Y(MM)
LOOP
CREATE A STATE CALLED MM
DISPLAY MM WITH "w1d.icn" AT (XM, YM)
CREATE A STATE CALLED MM
DISPLAY MM WITH "sd.icn" AT (XM, YM)
LET YM = LOCATION.Y(MM)-.H
WHILE S1 = 0 AND S2 = 0
DO
WAIT 1 UNIT
CREATE A STATE CALLED MM
DISPLAY MM WITH "s1.icn" AT (XM, YM)
LET XM = LOCATION.X(MM)+.L
LET YM = LOCATION.Y(MM)
IF TIME.V >= .CYCLE.TTIME
GO TO LABEL 1
ALWAYS
LOOP
CREATE A STATE CALLED MM
DISPLAY MM WITH "w1u.icn" AT (XM, YM)
CREATE A STATE CALLED MM
DISPLAY MM WITH "su.icn" AT (XM, YM)
LET YM = LOCATION.Y(MM)+.H
LOOP
END " END OF CM1

**D.3 Code for Control Signal of CM2**

**preamble.sim**

PREAMBLE
PROCESSES INCLUDE MM3, MM4, CM21, CM2
DEFINE S3, S4, S21 AS INTEGER VARIABLES
DEFINE ERROR, LABEL, REPORT AS A POINTER VARIABLE
DEFINE X, Y AS REAL VARIABLES
GRAPHIC ENTITIES INCLUDE STATE AND MESSAGE
DEFINE .L TO MEAN 25.0
DEFINE .H TO MEAN 50.0
" TIME WINDOWS FOR MM3
DEFINE .M3.S1 TO MEAN 2
DEFINE .M3.S2 TO MEAN 3
DEFINE .M3.S3 TO MEAN 1
Appendix D

DEFINE .M3.S4 TO MEAN 2
DEFINE .M3.S5 TO MEAN 3
DEFINE .M3.S6 TO MEAN 1
DEFINE .M3.S7 TO MEAN 2
DEFINE .M3.W1 TO MEAN 2
DEFINE .M3.W2 TO MEAN 3
DEFINE .M3.W3 TO MEAN 3
DEFINE .M3.W4 TO MEAN 2
DEFINE .M3.W5 TO MEAN 4
DEFINE .M3.W6 TO MEAN 3
DEFINE .M3.W7 TO MEAN 3
DEFINE .M3.W8 TO MEAN 5
* TIME WINDOWS FOR MM4
DEFINE .M4.S1 TO MEAN 1
DEFINE .M4.S2 TO MEAN 1
DEFINE .M4.S3 TO MEAN 3
DEFINE .M4.S4 TO MEAN 1
DEFINE .M4.S5 TO MEAN 2
DEFINE .M4.S6 TO MEAN 3
DEFINE .M4.W1 TO MEAN 9
DEFINE .M4.W2 TO MEAN 2
DEFINE .M4.W3 TO MEAN 2
DEFINE .M4.W4 TO MEAN 2
DEFINE .M4.W5 TO MEAN 6
DEFINE .M4.W6 TO MEAN 2
DEFINE .M4.W7 TO MEAN 4
* TIME WINDOWS FOR CM21
DEFINE .M21.S TO MEAN 2
DEFINE .M21.W1 TO MEAN 10
DEFINE .M21.W2 TO MEAN 1
DEFINE .M21.W3 TO MEAN 3
DEFINE .M21.W4 TO MEAN 3
DEFINE .M21.W5 TO MEAN 4
DEFINE .M21.W6 TO MEAN 3
DEFINE .M21.W7 TO MEAN 2
DEFINE .CYCLE.TIME TO MEAN 38
END * END OF PREAMBLE

main.sim

MAIN
LET X = 30
LET Y = 725
LET VXFORM.V = 1
LET TIMESCALE.V = 100
CALL SETWORLD.R(0.0, 1000.0, 0.0, 1000.0)
CREATE A MESSAGE CALLED REPORT
DISPLAY REPORT WITH "title.icn" AT (501, 900)
CREATE A MESSAGE CALLED REPORT
DISPLAY REPORT WITH "heading.icn" AT (501, 850)
CREATE A MESSAGE CALLED REPORT
DISPLAY REPORT WITH "legend.icn" AT (50, 275)
ACTIVATE A MM3 NOW
ACTIVATE A MM4 NOW
ACTIVATE A CM21 NOW
ACTIVATE A CM2 NOW
START SIMULATION
READ AS /
END * END OF MAIN

m3.sim

PROCESS MM3
DEFINE U, A31 AND A32 AS INTEGER VARIABLES
DEFINE M3 AS A POINTER VARIABLE
DEFINE X3, Y3 AS REAL VARIABLES
CREATE A MESSAGE CALLED LABEL
DISPLAY LABEL WITH "name3.icn" AT (10, Y+15)
LET S3 = 0
LET A31 = TIME.V
LET U = .M3.W1
CREATE A STATE CALLED M3
DISPLAY M3 WITH "w1d.icn" AT (X+U*.L, Y)
CREATE A STATE CALLED M3
DISPLAY M3 WITH "s2.icn" AT (X, Y)
CREATE A STATE CALLED M3
DISPLAY M3 WITH "sd.icn" AT (X+U*.L, Y)
WAIT U UNITS
LET A32 = TIME.V
IF (A32-A31) <= (U+U/10) AND (A32-A31) >= (U-U/10) " TIME WINDOW
   LET S3 = 1
   LET A31 = TIME.V
   LET U = .M3.S1
   LET X3 = LOCATION.X(M3)
   LET Y3 = LOCATION.Y(M3)
   CREATE A STATE CALLED M3
   DISPLAY M3 WITH "wlu.icn" AT (X3+U*.L, Y3-.H)
   CREATE A STATE CALLED M3
   DISPLAY M3 WITH "s3.icn" AT (X3, Y3+.H)
   CREATE A STATE CALLED M3
   DISPLAY M3 WITH "su.icn" AT (X3+U*.L, Y3+.H)
   WORK U UNITS
   LET A32 = TIME.V
ELSE
   IF(A32-A31) < (U-U/10) OR (A32-A31) > (U+U/10)
      CREATE A MESSAGE CALLED ERROR
      DISPLAY ERROR WITH "error3.icn" AT (50,275)
      READ AS /
      STOP
ALWAYS
IF (A32-A31) <= (U+U/10) AND (A32-A31) >= (U-U/10) " TIME WINDOW
   LET S3 = 0
   LET A31 = TIME.V
   LET U = .M3.W2
   LET X3 = LOCATION.X(M3)
   LET Y3 = LOCATION.Y(M3)
   CREATE A STATE CALLED M3
   DISPLAY M3 WITH "wlu.icn" AT (X3+U*.L, Y3+.H)
   CREATE A STATE CALLED M3
   DISPLAY M3 WITH "s3.icn" AT (X3, Y3+.H)
   CREATE A STATE CALLED M3
   DISPLAY M3 WITH "su.icn" AT (X3+U*.L, Y3+.H)
   WAIT U UNITS
   LET A32 = TIME.V
ELSE
   IF(A32-A31) < (U-U/10)
      CREATE A MESSAGE CALLED ERROR
      DISPLAY ERROR WITH "error3a.icn" AT (50,275)
      READ AS /
      STOP
ALWAYS
IF(A32-A31) > (U+U/10)
   CREATE A MESSAGE CALLED ERROR
   DISPLAY ERROR WITH "error3b.icn" AT (50,275)
   READ AS /
   STOP
ALWAYS
IF (A32-A31) <= (U+U/10) AND (A32-A31) >= (U-U/10) " TIME WINDOW
   LET S3 = 1
   LET A31 = TIME.V
Let U = M3.S2
Let X3 = LOCATION.X(M3)
Let Y3 = LOCATION.Y(M3)
Create a state called M3
Display M3 with "wld.icn" at (X3+U*.L, Y3+.H)
Create a state called M3
Display M3 with "s3.icn" at (X3, Y3+.H)
Create a state called M3
Display M3 with "sd.icn" at (X3+U*.L, Y3+.H)
Work U units
Let A32 = TIME.V

Else

If (A32-A31) < (U-U/10) OR (A32-A31) > (U-U/10)
Create a message called ERROR
Display ERROR with "errorc3.icn" at (50,275)
Read AS/
Always

Always
If (A32-A31) <= (U-U/10) AND (A32-A31) >= (U-U/10) "TIME WINDOW"
Let S3 = 0
Let A31 = TIME.V
Let U = M3.W3
Let X3 = LOCATION.X(M3)
Let Y3 = LOCATION.Y(M3)
Create a state called M3
Display M3 with "wlu.icn" at (X3+U*.L, Y3-.H)
Create a state called M3
Display M3 with "s2.icn" at (X3, Y3-.H)
Create a state called M3
Display M3 with "su.icn" at (X3+U*.L, Y3-.H)
Wait U units
Let A32 = TIME.V

Else

If (A32-A31) < (U-U/10)
Create a message called ERROR
Display ERROR with "error3a.icn" at (50,275)
Read AS/
Stop
Always
If (A32-A31) > (U-U/10)
Create a message called ERROR
Display ERROR with "error3b.icn" at (50,275)
Read AS/
Stop
Always

Always
If (A32-A31) <= (U-U/10) AND (A32-A31) >= (U-U/10) "TIME WINDOW"
Let S3 = 1
Let A31 = TIME.V
Let U = M3.S3
Let X3 = LOCATION.X(M3)
Let Y3 = LOCATION.Y(M3)
Create a state called M3
Display M3 with "wld.icn" at (X3+U*.L, Y3+.H)
Create a state called M3
Display M3 with "sl.icn" at (X3, Y3+.H)
Create a state called M3
Display M3 with "sd.icn" at (X3+U*.L, Y3+.H)
Work U units
Let A32 = TIME.V

Else

If (A32-A31) < (U-U/10) OR (A32-A31) > (U-U/10)
Create a message called ERROR
Display ERROR with "errorc3.icn" at (50,275)
Read AS/
Stop
Always

Always
If (A32-A31) <= (U-U/10) AND (A32-A31) >= (U-U/10) "TIME WINDOW"
LET S3 = 0
LET A31 = TIME.V
LET U = M3.W4
LET X3 = LOCATION.X(M3)
LET Y3 = LOCATION.Y(M3)
CREATE A STATE CALLED M3
DISPLAY M3 WITH "wlu.icn" AT (X3+U*.L, Y3-.H)
CREATE A STATE CALLED M3
DISPLAY M3 WITH "s2.icn" AT (X3, Y3-.H)
CREATE A STATE CALLED M3
DISPLAY M3 WITH "su.icn" AT (X3+U*.L, Y3-.H)
WAIT U UNITS
LET A32 = TIME.V
ELSE
    IF(A32-A31) < (U-U/10)
        CREATE A MESSAGE CALLED ERROR
        DISPLAY ERROR WITH "error3a.icn" AT (50,275)
        READ AS/
        STOP
    ALWAYS
    IF(A32-A31) > (U+U/10)
        CREATE A MESSAGE CALLED ERROR
        DISPLAY ERROR WITH "error3b.icn" AT (50,275)
        READ AS/
        STOP
    ALWAYS
    ALWAYS
    IF (A32-A31) <= (U+U/10) AND (A32-A31) >= (U-U/10) " TIME WINDOW
    LET S3 = 1
    LET A31 = TIME.V
    LET U = M3.S4
    LET X3 = LOCATION.X(M3)
    LET Y3 = LOCATION.Y(M3)
    CREATE A STATE CALLED M3
    DISPLAY M3 WITH "wld.icn" AT (X3+U*.L, Y3+.H)
    CREATE A STATE CALLED M3
    DISPLAY M3 WITH "s2.icn" AT (X3, Y3+.H)
    CREATE A STATE CALLED M3
    DISPLAY M3 WITH "sd.icn" AT (X3+U*.L, Y3+.H)
    WORK U UNITS
    LET A32 = TIME.V
ELSE
    IF(A32-A31) < (U-U/10) OR (A32-A31) > (U+U/10)
        CREATE A MESSAGE CALLED ERROR
        DISPLAY ERROR WITH "errorc3.icn" AT (50,275)
        READ AS/
        ALWAYS
    ALWAYS
    IF (A32-A31) <= (U+U/10) AND (A32-A31) >= (U-U/10) " TIME WINDOW
    LET S3 = 0
    LET A31 = TIME.V
    LET U = M3.W5
    LET X3 = LOCATION.X(M3)
    LET Y3 = LOCATION.Y(M3)
    CREATE A STATE CALLED M3
    DISPLAY M3 WITH "wld.icn" AT (X3+U*.L, Y3-.H)
    CREATE A STATE CALLED M3
    DISPLAY M3 WITH "s3.icn" AT (X3, Y3-.H)
    CREATE A STATE CALLED M3
    DISPLAY M3 WITH "sd.icn" AT (X3+U*.L, Y3-.H)
    WAIT U UNITS
    LET A32 = TIME.V
ELSE
    IF(A32-A31) < (U-U/10)
        CREATE A MESSAGE CALLED ERROR
        DISPLAY ERROR WITH "error3a.icn" AT (50,275)
        READ AS/
        STOP
    ALWAYS
IF(A32-A31) > (U+U/10)  
CREATE A MESSAGE CALLED ERROR  
DISPLAY ERROR WITH "error3b.icn" AT (50,275)  
READ AS/  
STOP  
ALWAYS

IF (A32-A31) <= (U+U/10) AND (A32-A31) >= (U-U/10) " TIME WINDOW
LET S3 = 1
LET A31 = TIME.V
LET X3 = LOCATION.X(M3)
LET Y3 = LOCATION.Y(M3)
CREATE A STATE CALLED M3
DISPLAY M3 WITH "wlu.icn" AT (X3+U*.L, Y3-.H)
CREATE A STATE CALLED M3
DISPLAY M3 WITH "s3.icn" AT (X3, Y3-.H)
CREATE A STATE CALLED M3
DISPLAY M3 WITH "su.icn" AT (X3+U*.L, Y3-.H)
WORK U UNITS
LET A32 = TIME.V
ELSE
IF(A32-A31) < (U-U/10) OR (A32-A31) > (U+U/10)  
CREATE A MESSAGE CALLED ERROR  
DISPLAY ERROR WITH "error3c.icn" AT (50,275)  
READ AS/  
STOP  
ALWAYS
IF (A32-A31) <= (U-U/10) AND (A32-A31) >= (U-U/10) " TIME WINDOW
LET S3 = 0
LET A31 = TIME.V
LET U = .M3.W6
LET X3 = LOCATION.X(M3)
LET Y3 = LOCATION.Y(M3)
CREATE A STATE CALLED M3
DISPLAY M3 WITH "wlu.icn" AT (X3+U*.L, Y3+.H)
CREATE A STATE CALLED M3
DISPLAY M3 WITH "s4.icn" AT (X3, Y3+.H)
CREATE A STATE CALLED M3
DISPLAY M3 WITH "su.icn" AT (X3+U*.L, Y3+.H)
WAIT U UNITS
LET A32 = TIME.V
ELSE
IF(A32-A31) < (U-U/10)  
CREATE A MESSAGE CALLED ERROR  
DISPLAY ERROR WITH "error3a.icn" AT (50,275)  
READ AS/  
STOP  
ALWAYS
IF(A32-A31) > (U+U/10)  
CREATE A MESSAGE CALLED ERROR  
DISPLAY ERROR WITH "error3b.icn" AT (50,275)  
READ AS/  
STOP  
ALWAYS
IF (A32-A31) <= (U+U/10) AND (A32-A31) >= (U-U/10) " TIME WINDOW
LET S3 = 1
LET A31 = TIME.V
LET U = .M3.S6
LET X3 = LOCATION.X(M3)
LET Y3 = LOCATION.Y(M3)
CREATE A STATE CALLED M3
DISPLAY M3 WITH "wld.icn" AT (X3+U*.L, Y3+.H)
CREATE A STATE CALLED M3
DISPLAY M3 WITH "sl.icn" AT (X3, Y3+.H)
CREATE A STATE CALLED M3
DISPLAY M3 WITH "sd.icn" AT (X3+U*.L, Y3+.H)
ELSE

IF \((A32-A31) < (U-U/10)\) OR \((A32-A31) > (U+U/10)\)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "errorc3.icn" AT (50, 275)
READ AS /
ALWAYS

ALWAYS

IF \((A32-A31) <= (U+U/10) AND (A32-A31) >= (U-U/10)\) " TIME WINDOW
LET S3 = 0
LET A31 = TIME.V
LET U = .M3.W7
LET X3 = LOCATION.X(M3)
LET Y3 = LOCATION.Y(M3)
CREATE A STATE CALLED M3
DISPLAY M3 WITH "wlu.icn" AT (X3+U*.L, Y3-.H)
CREATE A STATE CALLED M3
DISPLAY M3 WITH "su.icn" AT (X3+U*.L, Y3-.H)
WAIT U UNITS
LET A32 = TIME.V

ELSE

IF \((A32-A31) < (U-U/10)\)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "error3a.icn" AT (50, 275)
READ AS /
STOP
ALWAYS

IF \((A32-A31) > (U+U/10)\)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "error3b.icn" AT (50, 275)
READ AS /
STOP
ALWAYS

ALWAYS

IF \((A32-A31) <= (U+U/10) AND (A32-A31) >= (U-U/10)\) " TIME WINDOW
LET S3 = 1
LET A31 = TIME.V
LET U = .M3.S7
LET X3 = LOCATION.X(M3)
LET Y3 = LOCATION.Y(M3)
CREATE A STATE CALLED M3
DISPLAY M3 WITH "wld.icn" AT (X3+U*.L, Y3+.H)
CREATE A STATE CALLED M3
DISPLAY M3 WITH "sd.icn" AT (X3+U*.L, Y3+.H)
WORK U UNITS
LET A32 = TIME.V

ELSE

IF \((A32-A31) < (U-U/10)\) OR \((A32-A31) > (U+U/10)\)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "errorc3.icn" AT (50, 275)
READ AS /
STOP
ALWAYS

ALWAYS

IF \((A32-A31) <= (U+U/10) AND (A32-A31) >= (U-U/10)\) " TIME WINDOW
LET S3 = 0
LET A31 = TIME.V
LET U = .M3.W8
LET X3 = LOCATION.X(M3)
LET Y3 = LOCATION.Y(M3)
"CREATE A STATE CALLED M3"
"DISPLAY M3 WITH "wld.icn" AT (X3+U*.L, Y3-.H)"
CREATE A STATE CALLED M3
DISPLAY M3 WITH "s5.icn" AT (X3, Y3-.H)
"CREATE A STATE CALLED M3
"DISPLAY M3 WITH "sd.icn" AT (X3+U*.L, Y3-.H)
WAIT U UNITS
LET A32 = TIME.V
ELSE
    IF(A32-A31) < (U-U/10)
        CREATE A MESSAGE CALLED ERROR
        DISPLAY ERROR WITH "error3a.icn" AT (50, 275)
        READ AS/
        STOP
    ALWAYS
    IF(A32-A31) > (U+U/10)
        CREATE A MESSAGE CALLED ERROR
        DISPLAY ERROR WITH "error3b.icn" AT (50, 275)
        READ AS/
        STOP
    ALWAYS
    ALWAYS
    IF (A32-A31) <= (U+U/10) AND (A32-A31) >= (U-U/10)
        LET S3 = 0
    ELSE
        IF(A32-A31) < (U-U/10) OR (A32-A31) > (U+U/10)
            CREATE A MESSAGE CALLED ERROR
            DISPLAY ERROR WITH "erTorc3.icn"
            AT (50, 275)
            READ AS/
            STOP
    ALWAYS
    ALWAYS
    END  " END OF M3

m4.sim

PROCESS MM4
DEFINE U, A41 AND A42 AS INTEGER VARIABLES
DEFINE M4 AS A POINTER VARIABLE
CREATE A MESSAGE CALLED LABEL
DISPLAY LABEL WITH "name4.icn" AT (10, Y-125+15)
DEFINE X4, Y4 AS REAL VARIABLES
LET S4 = 0
LET A41 = TIME.V
LET U = .M4.W1
CREATE A STATE CALLED M4
DISPLAY M4 WITH "w1u.icn" AT (X+U*.L, Y-125)
CREATE A STATE CALLED M4
DISPLAY M4 WITH "s9.icn" AT (X, Y-125)
CREATE A STATE CALLED M4
DISPLAY M4 WITH "su.icn" AT (X+U*.L, Y-125)
CREATE A MESSAGE CALLED LABEL
DISPLAY LABEL WITH "line.icn" AT (X+U*.L, 800)
CREATE A MESSAGE CALLED LABEL
DISPLAY LABEL WITH "note.icn" AT (X+U*.L+60, 370)
WAIT U UNITS
LET A42 = TIME.V
IF (A42-A41) <= (U+U/10) AND (A42-A41) >= (U-U/10)  " TIME WINDOW
    LET S4 = 1
    LET A41 = TIME.V
    LET U = .M4.S1
    LET X4 = LOCATION.X(M4)
    LET Y4 = LOCATION.Y(M4)
    CREATE A STATE CALLED M4
    DISPLAY M4 WITH "wld.icn" AT (X4+U*.L, Y4+.H)
    CREATE A STATE CALLED M4
    DISPLAY M4 WITH "sl.icn" AT (X4, Y4+.H)
    CREATE A STATE CALLED M4
    DISPLAY M4 WITH "sd.icn" AT (X4+U*.L, Y4+.H)
WORK U UNITS
LET A42 = TIME.V
ELSE
IF (A42 - A41) < (U - U/10) OR (A42 - A41) > (U + U/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "error4.icn" AT (50, 275)
READ AS/
STOP
ALWAYS
IF (A42 - A41) <= (U + U/10) AND (A42 - A41) >= (U - U/10) "TIME WINDOW
LET S4 = 0
LET A41 = TIME.V
LET U = .M4.W2
LET X4 = LOCATION.X(M4)
LET Y4 = LOCATION.Y(M4)
CREATE A STATE CALLED M4
DISPLAY M4 WITH "wlu.icn" AT (X4 + U*.L, Y4-.H)
CREATE A STATE CALLED M4
DISPLAY M4 WITH "s2.icn" AT (X4, Y4-.H)
CREATE A STATE CALLED M4
DISPLAY M4 WITH "su.icn" AT (X4+U*.L, Y4-.H)
WAIT U UNITS
LET A42 = TIME.V
ELSE
IF (A42 - A41) LT (U - U/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "error4a.icn" AT (50, 275)
READ AS/
STOP
ALWAYS
IF (A42 - A41) GT (U + U/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "error4b.icn" AT (50, 275)
READ AS/
STOP
ALWAYS
IF (A42 - A41) <= (U + U/10) AND (A42 - A41) >= (U - U/10) "TIME WINDOW
LET S4 = 1
LET A41 = TIME.V
LET U = .M4.S2
LET X4 = LOCATION.X(M4)
LET Y4 = LOCATION.Y(M4)
CREATE A STATE CALLED M4
DISPLAY M4 WITH "wld.icn" AT (X4 + U*.L, Y4+.H)
CREATE A STATE CALLED M4
DISPLAY M4 WITH "si.ion" AT (X4, Y4+.H)
CREATE A STATE CALLED M4
DISPLAY M4 WITH "sd.icn" AT (X4 + U*.L, Y4+.H)
WORK U UNITS
LET A42 = TIME.V
ELSE
IF (A42 - A41) < (U - U/10) OR (A42 - A41) > (U + U/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "error4a.icn" AT (50, 275)
READ AS/
STOP
ALWAYS
IF (A42 - A41) <= (U + U/10) AND (A42 - A41) >= (U - U/10) "TIME WINDOW
LET S4 = 0
LET A41 = TIME.V
LET U = .M4.W3
LET X4 = LOCATION.X(M4)
LET Y4 = LOCATION.Y(M4)
CREATE A STATE CALLED M4
DISPLAY M4 WITH "wlu.icn" AT (X4 + U*.L, Y4-.H)
CREATE A STATE CALLED M4
DISPLAY M4 WITH "s2.icn" AT (X4, Y4-.H)
CREATE A STATE CALLED M4
DISPLAY M4 WITH "su.icn" AT (X4+U*.L, Y4-.H)
WAIT U UNITS
LET A42 = TIME.V
ELSE
IF (A42-A41) LT (U-U/10)
  CREATE A MESSAGE CALLED ERROR
  DISPLAY ERROR WITH "error4a.icn" AT (50, 275)
  READ AS/
  STOP
ALWAYS
IF (A42-A41) GT (U+U/10)
  CREATE A MESSAGE CALLED ERROR
  DISPLAY ERROR WITH "error4b.icn" AT (50, 275)
  READ AS/
  STOP
ALWAYS
ALWAYS
IF (A42-A41) <= (U+U/10) AND (A42-A41) >= (U-U/10) " TIME WINDOW
LET S4 = 1
LET A41 = TIME.V
LET U = .M4.S3
LET X4 = LOCATION.X(M4)
LET Y4 = LOCATION.Y(M4)
CREATE A STATE CALLED M4
DISPLAY M4 WITH "wd.icn" AT (X4+U*.L, Y4+.H)
CREATE A STATE CALLED M4
DISPLAY M4 WITH "s3.icn" AT (X4, Y4+.H)
CREATE A STATE CALLED M4
DISPLAY M4 WITH "sd.icn" AT (X4+U*.L, Y4+.H)
WORK U UNITS
LET A42 = TIME.V
ELSE
IF (A42-A41) < (U-U/10) OR (A42-A41) > (U+U/10)
  CREATE A MESSAGE CALLED ERROR
  DISPLAY ERROR WITH "error4a.icn" AT (50, 275)
  READ AS/
  ALWAYS
  STOP
ALWAYS
IF (A42-A41) <= (U+U/10) AND (A42-A41) >= (U-U/10) " TIME WINDOW
LET S4 = 0
LET A41 = TIME.V
LET U = .M4.W4
LET X4 = LOCATION.X(M4)
LET Y4 = LOCATION.Y(M4)
CREATE A STATE CALLED M4
DISPLAY M4 WITH "wlu.icn" AT (X4+U*.L, Y4-.H)
CREATE A STATE CALLED M4
DISPLAY M4 WITH "s2.icn" AT (X4, Y4-.H)
CREATE A STATE CALLED M4
DISPLAY M4 WITH "su.icn" AT (X4+U*.L, Y4-.H)
WAIT U UNITS
LET A42 = TIME.V
ELSE
IF (A42-A41) LT (U-U/10)
  CREATE A MESSAGE CALLED ERROR
  DISPLAY ERROR WITH "error4a.icn" AT (50, 275)
  READ AS/
  STOP
ALWAYS
IF (A42-A41) GT (U+U/10)
  CREATE A MESSAGE CALLED ERROR
  DISPLAY ERROR WITH "error4b.icn" AT (50, 275)
  READ AS/
  STOP
ALWAYS
ALWAYS
IF (A42-A41) <= (U+U/10) AND (A42-A41) >= (U-U/10) " TIME WINDOW
    LET S4 = 1
    LET A41 = TIME.V
    LET U = .M4.S4
    LET X4 = LOCATION.X(M4)
    LET Y4 = LOCATION.Y(M4)
    CREATE A STATE CALLED M4
    DISPLAY M4 WITH "wld.icn" AT (X4+U*.L, Y4+H)
    CREATE A STATE CALLED M4
    DISPLAY M4 WITH "s1.icn" AT (X4, Y4+H)
    CREATE A STATE CALLED M4
    DISPLAY M4 WITH "sd.icn" AT (X4+U*.L, Y4+H)
    CREATE A MESSAGE CALLED ERROR
    DISPLAY ERROR WITH "error4a.icn" AT (50, 275)
    READ AS/
    STOP
    ALWAYS

ALWAYS

IF (A42-A41) <= (U+U/10) AND (A42-A41) >= (U-U/10) " TIME WINDOW
    LET S4 = 0
    LET A41 = TIME.V
    LET U = .M4.W5
    LET X4 = LOCATION.X(M4)
    LET Y4 = LOCATION.Y(M4)
    CREATE A STATE CALLED M4
    DISPLAY M4 WITH "wld.icn" AT (X4+U*.L, Y4-H)
    CREATE A STATE CALLED M4
    DISPLAY M4 WITH "s6.icn" AT (X4, Y4-H)
    CREATE A STATE CALLED M4
    DISPLAY M4 WITH "sd.icn" AT (X4+U*.L, Y4-H)
    CREATE A MESSAGE CALLED LABEL
    DISPLAY LABEL WITH "line.icn" AT (X4+U*.L, 800)
    CREATE A MESSAGE CALLED LABEL
    DISPLAY LABEL WITH "note.icn" AT (X4+U*.L+5, 370)
    WAIT U UNITS
    LET A42 = TIME.V
    ELSE

IF (A42-A41) LT (U-U/10)
    CREATE A MESSAGE CALLED ERROR
    DISPLAY ERROR WITH "error4a.icn" AT (50, 275)
    READ AS/
    STOP
    ALWAYS

IF (A42-A41) GT (U+U/10)
    CREATE A MESSAGE CALLED ERROR
    DISPLAY ERROR WITH "error4b.icn" AT (50, 275)
    READ AS/
    STOP
    ALWAYS

ALWAYS

IF (A42-A41) <= (U+U/10) AND (A42-A41) >= (U-U/10) " TIME WINDOW
    LET S4 = 1
    LET A41 = TIME.V
    LET U = .M4.S5
    LET X4 = LOCATION.X(M4)
    LET Y4 = LOCATION.Y(M4)
    CREATE A STATE CALLED M4
    DISPLAY M4 WITH "wlu.icn" AT (X4+U*.L, Y4-H)
    CREATE A STATE CALLED M4
    DISPLAY M4 WITH "s2.icn" AT (X4, Y4-H)
    CREATE A STATE CALLED M4
    DISPLAY M4 WITH "su.icn" AT (X4+U*.L, Y4-H)
    CREATE A MESSAGE CALLED LABEL
    DISPLAY LABEL WITH "line.icn" AT (X4+U*.L, 800)
    CREATE A MESSAGE CALLED LABEL
    DISPLAY LABEL WITH "note.icn" AT (X4+U*.L+5, 370)
    WAIT U UNITS
    LET A42 = TIME.V
    ELSE
IF(A42-A41) < (U-U/10) OR (A42-A41) > (U+U/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "error4a.icn" AT (50,275)
READ AS /
STOP
ALWAYS

IF (A42-A41) <= (U+U/10) AND (A42-A41) >= (U-U/10) "TIME WINDOW
LET S4 = 0
LET A41 = TIME.V
LET U = M4.W6
LET X4 = LOCATION.X(M4)
LET Y4 = LOCATION.Y(M4)
CREATE A STATE CALLED M4
DISPLAY M4 WITH "wld.icn" AT (X4+U*.L, Y4+.H)
CREATE A STATE CALLED M4
DISPLAY M4 WITH "s2.icn" AT (X4, Y4+.H)
CREATE A STATE CALLED M4
DISPLAY M4 WITH "sd.icn" AT (X4+U*.L, Y4+.H)
WAIT U UNITS
LET A42 = TIME.V
ELSE
IF(A42-A41) LT (U-U/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "error4a.icn" AT (50,275)
READ AS /
STOP
ALWAYS
IF (A42-A41) GT (U+U/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "error4b.icn" AT (50,275)
READ AS /
STOP
ALWAYS

ALWAYS
ALWAYS
IF (A42-A41) <= (U+U/10) AND (A42-A41) >= (U-U/10) "TIME WINDOW
LET S4 = 1
LET A41 = TIME.V
LET U = M4.W6
LET X4 = LOCATION.X(M4)
LET Y4 = LOCATION.Y(M4)
CREATE A STATE CALLED M4
DISPLAY M4 WITH "wlu.icn" AT (X4+U*.L, Y4-.H)
CREATE A STATE CALLED M4
DISPLAY M4 WITH "s3.icn" AT (X4, Y4-.H)
CREATE A STATE CALLED M4
DISPLAY M4 WITH "su.icn" AT (X4+U*.L, Y4-.H)
WORK U UNITS
LET A42 = TIME.V
ELSE
IF(A42-A41) < (U-U/10) OR (A42-A41) > (U+U/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "error4a.icn" AT (50,275)
READ AS /
STOP
ALWAYS
ALWAYS
IF (A42-A41) <= (U+U/10) AND (A42-A41) >= (U-U/10) "TIME WINDOW
LET S4 = 0
LET A41 = TIME.V
LET U = M4.W7
LET X4 = LOCATION.X(M4)
LET Y4 = LOCATION.Y(M4)
CREATE A STATE CALLED M4
DISPLAY M4 WITH "s4.icn" AT (X4, Y4+.H)
WAIT U UNITS
LET A42 = TIME.V
ELSE
IF(A42-A41) LT (U-U/10)
Appendix D

create a message called error
display error with "error4a.icn" at (50, 275)
read as/
stop
always
if (a42-a41) gt (u+u/10)
create a message called error
display error with "error4b.icn" at (50, 275)
read as/
stop
always
if (a42-a41) <= (u+u/10) and (a42-a41) >= (u-u/10) " time window
let s4 = 0
else
if (a42-a41) < (u-u/10) or (a42-a41) > (u+u/10)
create a message called error
display error with "errorc4.icn" at (50, 275)
read as/
stop
always
always
end

cm21.sim

process cm21
define u, a211 and a212 as integer variables
define m21 as a pointer variable
create a message called label
display label with "namec21.icn" at (10, y-250+15)
define x21, y21 as real variables
let s21 = 0
let a211 = time.v
let u = .m21.w1
create a state called m21
display m21 with "wlu.icn" at (x+u*.l, y-250)
create a state called m21
display m21 with "sl0.icn" at (x, y-250)
create a state called m21
display m21 with "su.icn" at (x+u*.l, y-250)
wait u units
let a212 = time.v
if (a212-a211) <= (u+u/10) and (a212-a211) >= (u-u/10) " time window
let s21 = 1
let a211 = time.v
let u = .m21.s
let x21 = location.x(m21)
let y21 = location.y(m21)
create a state called m21
display m21 with "wld.icn" at (x21+u*.l, y21+.h)
create a state called m21
display m21 with "s2.icn" at (x21, y21+.h)
create a state called m21
display m21 with "sd.icn" at (x21+u*.l, y21+.h)
work u units
let a212 = time.v
else
if (a212-a211) < (u-u/10) or (a212-a211) > (u+u/10)
create a message called error
display error with "errorc21.icn" at (50, 275)
read as/
stop
always
always
always

Appendix D

IF (A212-A211) <= (U+U/10) AND (A212-A211) >= (U-U/10) "TIME WINDOW
   LET S21 = 0
   LET A211 = TIME.V
   LET U = .M21.W2
   LET X21 = LOCATION.X(M21)
   LET Y21 = LOCATION.Y(M21)
   CREATE A STATE CALLED M21
   DISPLAY M21 WITH "wlu.icn" AT (X21+U*.L, Y21-.H)
   CREATE A STATE CALLED M21
   DISPLAY M21 WITH "s1.icn" AT (X21, Y21-.H)
   CREATE A STATE CALLED M21
   DISPLAY M21 WITH "su.icn" AT (X21+U*.L, Y21-.H)
   WAIT U UNITS
   LET A212 = TIME.V
ELSE
   IF(A212-A211) LT (U-U/10)
      CREATE A MESSAGE CALLED ERROR
      DISPLAY ERROR WITH "error21a.icn" AT (50, 275)
      READ AS/
      STOP
   ALWAYS
   IF (A212-A211) GT (U+U/10)
      CREATE A MESSAGE CALLED ERROR
      DISPLAY ERROR WITH "error21b.icn" AT (50, 275)
      READ AS/
      STOP
   ALWAYS
END
ALWAYS
IF (A212-A211) <= (U+U/10) AND (A212-A211) >= (U-U/10) "TIME WINDOW
   LET S21 = 1
   LET A211 = TIME.V
   LET U = .M21.S
   LET X21 = LOCATION.X(M21)
   LET Y21 = LOCATION.Y(M21)
   CREATE A STATE CALLED M21
   DISPLAY M21 WITH "wld.icn" AT (X21+U*.L, Y21+.H)
   CREATE A STATE CALLED M21
   DISPLAY M21 WITH "s2.icn" AT (X21, Y21+.H)
   CREATE A STATE CALLED M21
   DISPLAY M21 WITH "sd.icn" AT (X21+U*.L, Y21+.H)
   WORK U UNITS
   LET A212 = TIME.V
ELSE
   IF(A212-A211) < (U-U/10) OR (A212-A211) > (U+U/10)
      CREATE A MESSAGE CALLED ERROR
      DISPLAY ERROR WITH "error21a.icn" AT (50,275)
      READ AS/
      STOP
   ALWAYS
END
ALWAYS
IF (A212-A211) <= (U+U/10) AND (A212-A211) >= (U-U/10) "TIME WINDOW
   LET S21 = 0
   LET A211 = TIME.V
   LET U = .M21.W3
   LET X21 = LOCATION.X(M21)
   LET Y21 = LOCATION.Y(M21)
   CREATE A STATE CALLED M21
   DISPLAY M21 WITH "wlu.icn" AT (X21+U*.L, Y21-.H)
   CREATE A STATE CALLED M21
   DISPLAY M21 WITH "s3.icn" AT (X21, Y21-.H)
   CREATE A STATE CALLED M21
   DISPLAY M21 WITH "su.icn" AT (X21+U*.L, Y21-.H)
   WAIT U UNITS
   LET A212 = TIME.V
ELSE
   IF(A212-A211) LT (U-U/10)
      CREATE A MESSAGE CALLED ERROR
      DISPLAY ERROR WITH "error21a.icn" AT (50, 275)
      READ AS/
      STOP

Appendix D

STOP
ALWAYS
IF (A212-A211) GT (U+U/10)
    CREATE A MESSAGE CALLED ERROR
    DISPLAY ERROR WITH "error21b.icn" AT (50, 275)
    READ AS /
STOP
ALWAYS

ALWAYS
IF (A212-A211) <= (U+U/10) AND (A212-A211) >= (U-U/10) " TIME WINDOW
    LET S21 = 1
    LET A211 = TIME.V
    LET U = .M21.S
    LET X21 = LOCATION.X(M21)
    LET Y21 = LOCATION.Y(M21)
    CREATE A STATE CALLED M21
    DISPLAY M21 WITH "wld.icn" AT (X21+U*.L, Y21+.H)
    CREATE A STATE CALLED M21
    DISPLAY M21 WITH "s2.icn" AT (X21, Y21+.H)
    CREATE A STATE CALLED M21
    DISPLAY M21 WITH "sd.icn" AT (X21+U*.L, Y21+.H)
    WORK U UNITS
    LET A212 = TIME.V
    ELSE
        IF (A212-A211) < (U-U/10) OR (A212-A211) > (U+U/10)
            CREATE A MESSAGE CALLED ERROR
            DISPLAY ERROR WITH "error21a.icn" AT (50, 275)
            READ AS /
            STOP
        ALWAYS
    ELSE
        IF (A212-A211) LT (U-U/10)
            CREATE A MESSAGE CALLED ERROR
            DISPLAY ERROR WITH "error21a.icn" AT (50, 275)
            READ AS /
        ALWAYS
ALWAYS
IF (A212-A211) <= (U+U/10) AND (A212-A211) >= (U-U/10) " TIME WINDOW
    LET S21 = 1
    LET A211 = TIME.V
    LET U = .M21.S
    LET X21 = LOCATION.X(M21)
    LET Y21 = LOCATION.Y(M21)
    CREATE A STATE CALLED M21
    DISPLAY M21 WITH "wld.icn" AT (X21+U*.L, Y21+.H)
    CREATE A STATE CALLED M21
    DISPLAY M21 WITH "s2.icn" AT (X21, Y21+.H)
    CREATE A STATE CALLED M21
    DISPLAY M21 WITH "sd.icn" AT (X21+U*.L, Y21+.H)
    WORK U UNITS
    LET A212 = TIME.V
    ELSE
        IF (A212-A211) < (U-U/10)
            CREATE A MESSAGE CALLED ERROR
            DISPLAY ERROR WITH "error21a.icn" AT (50, 275)
            READ AS /
            STOP
        ALWAYS
ALWAYS
IF (A212-A211) <= (U+U/10) AND (A212-A211) >= (U-U/10) " TIME WINDOW
    LET S21 = 1
    LET A211 = TIME.V
    LET U = .M21.S
    LET X21 = LOCATION.X(M21)
    LET Y21 = LOCATION.Y(M21)
    CREATE A STATE CALLED M21
    DISPLAY M21 WITH "wld.icn" AT (X21+U*.L, Y21+.H)
    CREATE A STATE CALLED M21
    DISPLAY M21 WITH "s2.icn" AT (X21, Y21+.H)
CREATE A STATE CALLED M21
DISPLAY M21 WITH "sd.icn" AT (X21+U*.L, Y21+H)
CREATE A MESSAGE CALLED LABEL
DISPLAY LABEL WITH "line.icn" AT (X21+U*.L, 800)
WORK U UNITS
LET A212 = TIME.V
ELSE
IF((A212-A211) < (U-U/10) OR (A212-A211) > (U+U/10))
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "error21a.icn" AT (50, 275)
READ AS/
STOP
ALWAYS
IF (A212-A211) <= (U+U/10) AND (A212-A211) >= (U-U/10) " TIME WINDOW
LET S21 = 0
LET A211 = TIME.V
LET U = .M21.W5
LET X21 = LOCATION.X(M21)
LET Y21 = LOCATION.Y(M21)
CREATE A STATE CALLED M21
DISPLAY M21 WITH "wlu.icn" AT (X21+U*.L, Y21-H)
CREATE A STATE CALLED M21
DISPLAY M21 WITH "s4.icn" AT (X21, Y21-H)
CREATE A STATE CALLED M21
DISPLAY M21 WITH "su.icn" AT (X21+U*.L, Y21-H)
WAIT U UNITS
LET A212 = TIME.V
IF(A212-A211) LT (U-U/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "error21a.icn" AT (50, 275)
READ AS/
STOP
ALWAYS
IF (A212-A211) GT (U+U/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "error21b.icn" AT (50, 275)
READ AS/
STOP
ALWAYS
ELSE
IF((A212-A211) < (U-U/10) OR (A212-A211) > (U+U/10))
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "error21a.icn" AT (50, 275)
READ AS/
STOP
ALWAYS
IF (A212-A211) <= (U+U/10) AND (A212-A211) >= (U-U/10) " TIME WINDOW
LET S21 = 1
LET A211 = TIME.V
LET U = .M21.S
LET X21 = LOCATION.X(M21)
LET Y21 = LOCATION.Y(M21)
CREATE A STATE CALLED M21
DISPLAY M21 WITH "wld.icn" AT (X21+U*D, Y21+H)
CREATE A STATE CALLED M21
DISPLAY M21 WITH "s2.icn" AT (X21, Y21+H)
CREATE A STATE CALLED M21
DISPLAY M21 WITH "sd.icn" AT (X21+U*.L, Y21+H)
WAIT U UNITS
LET A212 = TIME.V
IF(A212-A211) LT (U-U/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "error21a.icn" AT (50, 275)
READ AS/
STOP
ALWAYS
IF (A212-A211) GT (U+U/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "error21b.icn" AT (50, 275)
READ AS/
STOP
ALWAYS
ELSE
IF((A212-A211) < (U-U/10) OR (A212-A211) > (U+U/10))
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "error21a.icn" AT (50, 275)
READ AS/
STOP
ALWAYS
IF (A212-A211) <= (U+U/10) AND (A212-A211) >= (U-U/10) " TIME WINDOW
LET S21 = 0
LET A211 = TIME.V
LET U = .M21.W6
LET X21 = LOCATION.X(M21)
LET Y21 = LOCATION.Y(M21)
CREATE A STATE CALLED M21
DISPLAY M21 WITH "wlu.icn" AT (X21+U*L, Y21-H)
CREATE A STATE CALLED M21
DISPLAY M21 WITH "s3.icn" AT (X21, Y21-H)
CREATE A STATE CALLED M21
DISPLAY M21 WITH "su.icn" AT (X21+U*L, Y21-H)
WAIT U UNITS
LET A212 = TIME.V
ELSE
IF (A212-A211) LT (U-U/10)
    CREATE A MESSAGE CALLED ERROR
    DISPLAY ERROR WITH "error21a.icn" AT (50,275)
    READ AS/
    STOP
ALWAYS
IF (A212-A211) GT (U+U/10)
    CREATE A MESSAGE CALLED ERROR
    DISPLAY ERROR WITH "error21b.icn" AT (50, 275)
    READ AS/
    STOP
ALWAYS
ALWAYS
IF (A212-A211) <= (U+U/10) AND (A212-A211) >= (U-U/10) " TIME WINDOW
    LET S21 = 1
    LET A211 = TIME.V
    LET U = .M21.S
    LET X21 = LOCATION.X(M21)
    LET Y21 = LOCATION.Y(M21)
    CREATE A STATE CALLED M21
    DISPLAY M21 WITH "wld.icn" AT (X21+U*L, Y21+H)
    CREATE A STATE CALLED M21
    DISPLAY M21 WITH "s2.icn" AT (X21, Y21+H)
    CREATE A STATE CALLED M21
    DISPLAY M21 WITH "sd.icn" AT (X21+U*L, Y21+H)
    CREATE A MESSAGE CALLED LABEL
    DISPLAY LABEL WITH "line.icn" AT (X21+U*L, 800)
    WORK U UNITS
    LET A212 = TIME.V
ELSE
IF (A212-A211) < (U-U/10) OR (A212-A211) > (U+U/10)
    CREATE A MESSAGE CALLED ERROR
    DISPLAY ERROR WITH "error21a.icn" AT (50,275)
    READ AS/
    STOP
ALWAYS
ALWAYS
IF (A212-A211) <= (U+U/10) AND (A212-A211) >= (U-U/10) " TIME WINDOW
    LET S21 = 0
    LET A211 = TIME.V
    LET U = .M21.W7
    LET X21 = LOCATION.X(M21)
    LET Y21 = LOCATION.Y(M21)
    CREATE A STATE CALLED M21
    DISPLAY M21 WITH "s2.icn" AT (X21, Y21-H)
    WAIT U UNITS
    LET A212 = TIME.V
ELSE
IF (A212-A211) LT (U-U/10)
    CREATE A MESSAGE CALLED ERROR
    DISPLAY ERROR WITH "error21a.icn" AT (50, 275)
    READ AS/
    STOP
ALWAYS
IF (A212-A211) GT (U+U/10)
    CREATE A MESSAGE CALLED ERROR
    DISPLAY ERROR WITH "error21b.icn" AT (50, 275)
    READ AS/
    STOP
ALWAYS
IF (A212-A211) <= (U+U/10) AND (A212-A211) >= (U-U/10)  " TIME WINDOW
LET S21 = 0
ELSE
IF (A212-A211) < (U-U/10) OR (A212-A211) > (U+U/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "errorc21.icn" AT (50,275)
READ AS /
STOP
ALWAYS
END  " END OF CM21

cm2.sim
PROCESS CM2
DEFINE MM2 AS A POINTER VARIABLE
CREATE A MESSAGE CALLED LABEL
DISPLAY LABEL WITH "namec2.icn" AT (10, Y-375+65)
DEFINE XM, YM AS REAL VARIABLES
LET XM = X
LET YM = Y-375.0
 LABEL1' WHILE TIME.V LT .CYCLE.TIME
DO
UNTIL S3 = 1 OR S4 = 1 OR S21 = 1
DO
WAIT 1 UNIT
CREATE A STATE CALLED MM2
DISPLAY MM2 WITH "s1.icn" AT (XM, YM)
LET XM = LOCATION.X(MM2)+1
LET YM = LOCATION.Y(MM2)
IF TIME.V >= .CYCLE.TIME
GO TO LABEL1
ALWAYS
LOOP
CREATE A STATE CALLED MM2
DISPLAY MM2 WITH "w2u.icn" AT (XM, YM)
CREATE A STATE CALLED MM2
DISPLAY MM2 WITH "su.icn" AT (XM, YM)
LET YM = LOCATION.Y(MM2)+H
UNTIL S3 = 0 AND S4 = 0 AND S21 = 0
DO
WAIT 1 UNIT
CREATE A STATE CALLED MM2
DISPLAY MM2 WITH "s1.icn" AT (XM, YM)
LET XM = LOCATION.X(MM2)+1
LET YM = LOCATION.Y(MM2)
LOOP
CREATE A STATE CALLED MM2
DISPLAY MM2 WITH "w2d.icn" AT (XM, YM)
CREATE A STATE CALLED MM2
DISPLAY MM2 WITH "sd.icn" AT (XM, YM)
LET YM = LOCATION.Y(MM2)-H
LOOP
END  " END OF CM2

D.4 Code for Control Signal of CM21

preamble.sim
PREAMBLE
PROCESSES INCLUDE MM5, MM6, CM21
DEFINE S5 AND S6 AS INTEGER VARIABLES
Define error, label, report as a pointer variable
Define x, y as real variables
Graphic entities include state and message
Define l to mean 12.5
Define h to mean 50.0
Define m5.s to mean 1
Define m5.w1 to mean 20
Define m5.w to mean 2
Define m5.w2 to mean 6
Define m5.w3 to mean 8
Define m5.wl to mean 4
Define m6.s to mean 2
Define m6.w1 to mean 21
Define m6.w2 to mean 4
Define m6.w3 to mean 8
Define m6.w4 to mean 10
Define m6.w5 to mean 5
Define cycle.time to mean 76
End "end of preamble

Main.sim

Main
Let x = 30
Let y = 700
Let vxform.v = 1
Let timescale.v = 100
Call setworld.r(0.0, 1000.0, 0.0, 1000.0)
Create a message called report
Display report with "title.icn" at (501,900)
Create a message called report
Display report with "heading.icn" at (501,850)
Activate a mm5 now
Activate a mm6 now
Activate a cm21 now

Start simulation
Create a message called report
Display report with "legend.icn" at (50,275)
Read as/
End "end of main

m5.sim

Process mm5
Define u, a51 and a52 as integer variables
Define m5 as a pointer variable
Define x5, y5 as real variables
Create a message called label
Display label with "name5.icn" at (30, y+25)
Let s5 = 0
Let a51 = time.v
Let u = .m5.w1
Create a state called m5
Display m5 with "wlu.icn" at (x+u*l, y)
Create a state called m5
Display m5 with "s20.icn" at (x, y)
Create a state called m5
Display m5 with "su.icn" at (x+u*l, y)
Wait u units
Let a52 = time.v
If (a52-a51) <= (u+u/10) and (a52-a51) >= (u-u/10) "time window
Let s5 = 1
Let a51 = time.v
Let u = .m5.s
LET X5 = LOCATION.X(M5)
LET Y5 = LOCATION.Y(M5)
CREATE A STATE CALLED M5
DISPLAY M5 WITH "wld.icn" AT (X5+U*.L, Y5+.H)
CREATE A STATE CALLED M5
DISPLAY M5 WITH "s1.icn" AT (X5, Y5+.H)
CREATE A STATE CALLED M5
DISPLAY M5 WITH "sd.icn" AT (X5+U*.L, Y5+.H)
WORK U UNITS
LET A52 = TIME.V
ELSE
IF (A52-A51) < (U-U/10) OR (A52-A51) > (U+U/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "errorc5.icn" AT (50,275)
READ AS/
STOP
ALWAYS
IF (A52-A51) <= (U-U/10) AND (A52-A51) >= (U-U/10)
LET S5 = 0
LET A51 = TIME.V
LET U = .M5.W
LET X5 = LOCATION.X(M5)
LET Y5 = LOCATION.Y(M5)
CREATE A STATE CALLED M5
DISPLAY M5 WITH "wld.icn" AT (X5+U*.L, Y5-.H)
CREATE A STATE CALLED M5
DISPLAY M5 WITH "s2.icn" AT (X5, Y5-.H)
CREATE A STATE CALLED M5
DISPLAY M5 WITH "sd.icn" AT (X5+U*.L, Y5-.H)
WAIT U UNITS
LET A52 = TIME.V
ELSE
IF (A52-A51) < (U-U/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "error5a.icn" AT (50,275)
READ AS/
STOP
ALWAYS
IF (A52-A51) > (U+U/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "error5b.icn" AT (50,275)
READ AS/
STOP
ALWAYS
IF (A52-A51) <= (U-U/10) AND (A52-A51) >= (U-U/10)
LET S5 = 1
LET A51 = TIME.V
LET U = .M5.S
LET X5 = LOCATION.X(M5)
LET Y5 = LOCATION.Y(M5)
CREATE A STATE CALLED M5
DISPLAY M5 WITH "wlu.icn" AT (X5+U*.L, Y5-.H)
CREATE A STATE CALLED M5
DISPLAY M5 WITH "sl.icn" AT (X5, Y5-.H)
CREATE A STATE CALLED M5
DISPLAY M5 WITH "su.icn" AT (X5+U*.L, Y5-.H)
WORK U UNITS
LET A52 = TIME.V
ELSE
IF (A52-A51) < (U-U/10) OR (A52-A51) > (U+U/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "errorc5.icn" AT (50,275)
READ AS/
STOP
ALWAYS
IF (A52-A51) <= (U-U/10) AND (A52-A51) >= (U-U/10)
LET S5 = 0
"TIME WINDOW
LET A51 = TIME.V
LET U = .M5.W
LET X5 = LOCATION.X(M5)
LET Y5 = LOCATION.Y(M5)
CREATE A STATE CALLED M5
DISPLAY M5 WITH "wlu.icn" AT (X5+U*.L, Y5+.H)
CREATE A STATE CALLED M5
DISPLAY M5 WITH "s2.icn" AT (X5, Y5+.H)
CREATE A STATE CALLED M5
DISPLAY M5 WITH "su.icn" AT (X5+U*.L, Y5+.H)
WAIT U UNITS
LET A52 = TIME.V
ELSE
IF(A52-A51) < (U-U/10)
  CREATE A MESSAGE CALLED ERROR
  DISPLAY ERROR WITH "error5a.icn" AT (50, 275)
  READ AS/
  STOP
ALWAYS
IF(A52-A51) > (U+U/10)
  CREATE A MESSAGE CALLED ERROR
  DISPLAY ERROR WITH "error5b.icn" AT (50, 275)
  READ AS/
  STOP
ALWAYS
ALWAYS
IF (A52-A51) <= (U+U/10) AND (A52-A51) >= (U-U/10) " TIME WINDOW
LET S5 = 1
LET A51 = TIME.V
LET U = .M5.S
LET X5 = LOCATION.X(M5)
LET Y5 = LOCATION.Y(M5)
CREATE A STATE CALLED M5
DISPLAY M5 WITH "wld.icn" AT (X5+U*.L, Y5+.H)
CREATE A STATE CALLED M5
DISPLAY M5 WITH "sl.icn" AT (X5, Y5+.H)
CREATE A STATE CALLED M5
DISPLAY M5 WITH "sd.icn" AT (X5+U*.L, Y5+.H)
WORK U UNITS
LET A52 = TIME.V
ELSE
IF(A52-A51) < (U-U/10) OR (A52-A51) > (U+U/10)
  CREATE A MESSAGE CALLED ERROR
  DISPLAY ERROR WITH "errorc5.icn" AT (50,275)
  READ AS/
  STOP
ALWAYS
ALWAYS
IF (A52-A51) <= (U+U/10) AND (A52-A51) >= (U-U/10) " TIME WINDOW
LET S5 = 0
LET A51 = TIME.V
LET U = .M5.W
LET X5 = LOCATION.X(M5)
LET Y5 = LOCATION.Y(M5)
CREATE A STATE CALLED M5
DISPLAY M5 WITH "wld.icn" AT (X5+U*.L, Y5-.H)
CREATE A STATE CALLED M5
DISPLAY M5 WITH "sl.icn" AT (X5, Y5-.H)
CREATE A STATE CALLED M5
DISPLAY M5 WITH "sd.icn" AT (X5+U*.L, Y5-.H)
WAIT U UNITS
LET A52 = TIME.V
ELSE
IF(A52-A51) < (U-U/10)
  CREATE A MESSAGE CALLED ERROR
  DISPLAY ERROR WITH "error5a.icn" AT (50,275)
  READ AS/
  STOP
ALWAYS
IF(A52-A51) > (U+U/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "error5b.icn" AT (50,275)
READ AS/
STOP
ALWAYS

IF (A52-A51) <= (U+U/10) AND (A52-A51) >= (U-U/10)
LET S5 = 1
LET A51 = TIME.V
LET U = .M5.S
LET X5 = LOCATION.X(M5)
LET Y5 = LOCATION.Y(M5)
CREATE A STATE CALLED M5
DISPLAY M5 WITH "wlu.icn" AT (X5+U*.L, Y5-.H)
CREATE A STATE CALLED M5
DISPLAY M5 WITH "sl.icn" AT (X5, Y5-.H)
CREATE A STATE CALLED M5
DISPLAY M5 WITH "su.icn" AT (X5+U*.L, Y5-.H)
WORK U UNITS
LET A52 = TIME.V

ELSE
IF(A52-A51) < (U-U/10) OR (A52-A51) > (U+U/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "errorc5.icn" AT (50,275)
READ AS/
ALWAYS

IF (A52-A51) <= (U+U/10) AND (A52-A51) >= (U-U/10)
LET S5 = 0
LET A51 = TIME.V
LET U = .M5.W2
LET X5 = LOCATION.X(M5)
LET Y5 = LOCATION.Y(M5)
CREATE A STATE CALLED M5
DISPLAY M5 WITH "wlu.icn" AT (X5+U*.L, Y5+.H)
CREATE A STATE CALLED M5
DISPLAY M5 WITH "sl.icn" AT (X5, Y5+.H)
CREATE A STATE CALLED M5
DISPLAY M5 WITH "su.icn" AT (X5+U*.L, Y5+.H)
WAIT U UNITS
LET A52 = TIME.V

ELSE
IF(A52-A51) < (U-U/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "error5a.icn" AT (50,275)
READ AS/
STOP
ALWAYS
IF(A52-A51) > (U+U/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "error5b.icn" AT (50,275)
READ AS/
STOP
ALWAYS

ALWAYS
IF (A52-A51) <= (U+U/10) AND (A52-A51) >= (U-U/10)
LET S5 = 1
LET A51 = TIME.V
LET U = .M5.S
LET X5 = LOCATION.X(M5)
LET Y5 = LOCATION.Y(M5)
CREATE A STATE CALLED M5
DISPLAY M5 WITH "wld.icn" AT (X5+U*.L, Y5+.H)
CREATE A STATE CALLED M5
DISPLAY M5 WITH "sl.icn" AT (X5, Y5+.H)
CREATE A STATE CALLED M5
DISPLAY M5 WITH "sd.icn" AT (X5+U*.L, Y5+.H)
WORK U UNITS
Appendix D

LET A52 = TIME.V
IF (A52 - A51) < (U - U/10) OR (A52 - A51) > (U + U/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "error5a.icn" AT (50, 275)
READ AS /
STOP
ALWAYS

ALWAYS
IF (A52 - A51) <= (U + U/10) AND (A52 - A51) >= (U - U/10) " TIME WINDOW
LET S5 = 0
LET A51 = TIME.V
LET U = .M5.W
LET X5 = LOCATION.X(M5)
LET Y5 = LOCATION.Y(M5)
CREATE A STATE CALLED M5
DISPLAY M5 WITH "wld.icn" AT (X5 + U*JL, Y5 - H)
CREATE A STATE CALLED M5
DISPLAY M5 WITH "s2.icn" AT (X5, Y5 - H)
CREATE A STATE CALLED M5
DISPLAY M5 WITH "sd.icn" AT (X5 + U*.L, Y5 - H)
WAIT U UNITS
LET A52 = TIME.V

ELSE
IF (A52 - A51) < (U - U/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "error5a.icn" AT (50, 275)
READ AS /
STOP
ALWAYS
IF (A52 - A51) > (U + U/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "error5b.icn" AT (50, 275)
READ AS /
STOP
ALWAYS

ALWAYS
IF (A52 - A51) <= (U + U/10) AND (A52 - A51) >= (U - U/10) " TIME WINDOW
LET S5 = 1
LET A51 = TIME.V
LET U = .M5.S
LET X5 = LOCATION.X(M5)
LET Y5 = LOCATION.Y(M5)
CREATE A STATE CALLED M5
DISPLAY M5 WITH "wlu.icn" AT (X5 + U*E, Y5 - H)
CREATE A STATE CALLED M5
DISPLAY M5 WITH "sl.icn" AT (X5, Y5 - H)
CREATE A STATE CALLED M5
DISPLAY M5 WITH "su.icn" AT (X5 + U*.L, Y5 - H)
WORK U UNITS
LET A52 = TIME.V

ELSE
IF (A52 - A51) < (U - U/10) OR (A52 - A51) > (U + U/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "errorc5.icn" AT (50, 275)
READ AS /
STOP
ALWAYS

ALWAYS
IF (A52 - A51) <= (U + U/10) AND (A52 - A51) >= (U - U/10) " TIME WINDOW
LET S5 = 0
LET A51 = TIME.V
LET U = .M5.W2
LET X5 = LOCATION.X(M5)
LET Y5 = LOCATION.Y(M5)
CREATE A STATE CALLED M5
DISPLAY M5 WITH "wlu.icn" AT (X5 + U*.L, Y5+.H)
CREATE A STATE CALLED M5
DISPLAY M5 WITH "sl.icn" AT (X5, Y5+.H)
CREATE A STATE CALLED M5
DISPLAY M5 WITH "su.icn" AT (X5+U*.L, Y5+.H)
WAIT U UNITS
LET A52 = TIME.V

ELSE
IF(A52-A51) < (U-U/10)
    CREATE A MESSAGE CALLED ERROR
    DISPLAY ERROR WITH "error5a.icn" AT (50, 275)
    READ AS /
    STOP
ALWAYS
IF(A52-A51) > (U+U/10)
    CREATE A MESSAGE CALLED ERROR
    DISPLAY ERROR WITH "error5b.icn" AT (50, 275)
    READ AS /
    STOP
ALWAYS
IF (A52-A51) <= (U+U/10) AND (A52-A51) >= (U-U/10) " TIME WINDOW
    LET S5 = 1
    LET A51 = TIME.V
    LET U = .M5.S
    LET X5 = LOCATION.X(M5)
    LET Y5 = LOCATION.Y(M5)
    CREATE A STATE CALLED M5
    DISPLAY M5 WITH "wld.icn" AT (X5+U*.L, Y5+.H)
    CREATE A STATE CALLED M5
    DISPLAY M5 WITH "sl.icn" AT (X5, Y5+.H)
    CREATE A STATE CALLED M5
    DISPLAY M5 WITH "sd.icn" AT (X5+U*.L, Y5+.H)
    WORK U UNITS
    LET A52 = TIME.V
ELSE
IF(A52-A51) < (U-U/10) OR (A52-A51) > (U+U/10)
    CREATE A MESSAGE CALLED ERROR
    DISPLAY ERROR WITH "errorc5.icn" AT (50, 275)
    READ AS /
    STOP
ALWAYS
IF (A52-A51) <= (U+U/10) AND (A52-A51) >= (U-U/10) " TIME WINDOW
    LET S5 = 0
    LET A51 = TIME.V
    LET U = .M5.W
    LET X5 = LOCATION.X(M5)
    LET Y5 = LOCATION.Y(M5)
    CREATE A STATE CALLED M5
    DISPLAY M5 WITH "wld.icn" AT (X5+U*.L, Y5-.H)
    CREATE A STATE CALLED M5
    DISPLAY M5 WITH "s2.icn" AT (X5, Y5-.H)
    CREATE A STATE CALLED M5
    DISPLAY M5 WITH "sd.icn" AT (X5+U*.L, Y5-.H)
    WAIT U UNITS
    LET A52 = TIME.V
ELSE
IF(A52-A51) < (U-U/10)
    CREATE A MESSAGE CALLED ERROR
    DISPLAY ERROR WITH "error5a.icn" AT (50, 275)
    READ AS /
    STOP
ALWAYS
IF(A52-A51) > (U+U/10)
    CREATE A MESSAGE CALLED ERROR
    DISPLAY ERROR WITH "error5b.icn" AT (50, 275)
    READ AS /
    STOP
ALWAYS
IF (A52-A51) <= (U+U/10) AND (A52-A51) >= (U-U/10) " TIME WINDOW
    LET S5 = 1
LET A51 = TIME.V
LET U = .M5.S
LET X5 = LOCATION.X(M5)
LET Y5 = LOCATION.Y(M5)
CREATE A STATE CALLED M5
DISPLAY M5 WITH "wlu.icn" AT (X5+U*.L, Y5-.H)
CREATE A STATE CALLED M5
DISPLAY M5 WITH "sl.icn" AT (X5, Y5-.H)
CREATE A STATE CALLED M5
DISPLAY M5 WITH "su.icn" AT (X5+U*.L, Y5-.H)
WORK U UNITS
LET A52 = TIME.V
ELSE
IF(A52-A51) < (U-U/10) OR (A52-A51) > (U+U/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "errorc5.icn" AT (50,275)
READ AS/
ALWAYS
ALWAYS
IF (A52-A51) <= (U+U/10) AND (A52-A51) >= (U-U/10) " TIME WINDOW
LET S5 = 0
LET A51 = TIME.V
LET U = .M5.W3
LET X5 = LOCATION.X(M5)
LET Y5 = LOCATION.Y(M5)
CREATE A STATE CALLED M5
DISPLAY M5 WITH "wlu.icn" AT (X5+U*.L, Y5+.H)
CREATE A STATE CALLED M5
DISPLAY M5 WITH "sl.icn" AT (X5, Y5+.H)
CREATE A STATE CALLED M5
DISPLAY M5 WITH "su.icn" AT (X5+U*.L, Y5+.H)
WAIT U UNITS
LET A52 = TIME.V
ELSE
IF(A52-A51) < (U-U/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "error5a.icn" AT (50,275)
READ AS/
STOP
ALWAYS
IF(A52-A51) > (U+U/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "error5b.icn" AT (50,275)
READ AS/
STOP
ALWAYS
ALWAYS
IF (A52-A51) <= (U+U/10) AND (A52-A51) >= (U-U/10) " TIME WINDOW
LET S5 = 1
LET A51 = TIME.V
LET U = .M5.S
LET X5 = LOCATION.X(M5)
LET Y5 = LOCATION.Y(M5)
CREATE A STATE CALLED M5
DISPLAY M5 WITH "wld.icn" AT (X5+U*.L, Y5+.H)
CREATE A STATE CALLED M5
DISPLAY M5 WITH "sl.icn" AT (X5, Y5+.H)
CREATE A STATE CALLED M5
DISPLAY M5 WITH "sd.icn" AT (X5+U*.L, Y5+.H)
WORK U UNITS
LET A52 = TIME.V
ELSE
IF(A52-A51) < (U-U/10) OR (A52-A51) > (U+U/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "errorc5.icn" AT (50,275)
READ AS/
STOP
ALWAYS
ALWAYS
IF $(A52 - A51) <= (U + U/10)$ AND $(A52 - A51) >= (U - U/10)$ \quad \text{"TIME WINDOW"}

\begin{align*}
& \text{LET } S5 = 0 \\
& \text{LET } A51 = \text{TIME.V} \\
& \text{LET } U = M5.W \\
& \text{LET } X5 = \text{LOCATION.X(M5)} \\
& \text{LET } Y5 = \text{LOCATION.Y(M5)} \\
& \text{CREATE A STATE CALLED M5} \\
& \text{DISPLAY M5 WITH "wld.icn" AT } \langle X5+U*.L, Y5-.H \rangle \\
& \text{CREATE A STATE CALLED M5} \\
& \text{DISPLAY M5 WITH "s2.icn" AT } \langle X5, Y5-.H \rangle \\
& \text{CREATE A STATE CALLED M5} \\
& \text{DISPLAY M5 WITH "sd.icn" AT } \langle X5+U*.L, Y5-.H \rangle \\
& \text{WAIT } U \text{ UNITS} \\
& \text{LET } A52 = \text{TIME.V} \\
\end{align*}

ELSE

\begin{align*}
& \text{IF}(A52 - A51) < (U - U/10) \\
& \text{CREATE A MESSAGE CALLED ERROR} \\
& \text{DISPLAY ERROR WITH "error5a.icn" AT } \langle 50, 275 \rangle \\
& \text{READ AS/} \\
& \text{STOP} \\
& \text{ALWAYS} \\
& \text{IF}(A52 - A51) > (U + U/10) \\
& \text{CREATE A MESSAGE CALLED ERROR} \\
& \text{DISPLAY ERROR WITH "error5b.icn" AT } \langle 50, 275 \rangle \\
& \text{READ AS/} \\
& \text{STOP} \\
& \text{ALWAYS} \\
\end{align*}

ALWAYS

\begin{align*}
& \text{IF}(A52 - A51) <= (U + U/10) \text{ AND } (A52 - A51) >= (U - U/10) \quad \text{"TIME WINDOW"} \\
& \text{LET } S5 = 1 \\
& \text{LET } A51 = \text{TIME.V} \\
& \text{LET } U = .M5.W2 \\
& \text{LET } X5 = \text{LOCATION.X(M5)} \\
& \text{LET } Y5 = \text{LOCATION.Y(M5)} \\
& \text{CREATE A STATE CALLED M5} \\
& \text{DISPLAY M5 WITH "wlu.icn" AT } \langle X5+U*.L, Y5+.H \rangle \\
& \text{CREATE A STATE CALLED M5} \\
& \text{DISPLAY M5 WITH "sl.icn" AT } \langle X5, Y5+.H \rangle \\
& \text{CREATE A STATE CALLED M5} \\
& \text{DISPLAY M5 WITH "su.icn" AT } \langle X5+U*.L, Y5+.H \rangle \\
& \text{WORK } U \text{ UNITS} \\
& \text{LET } A52 = \text{TIME.V} \\
\end{align*}

ELSE

\begin{align*}
& \text{IF}(A52 - A51) < (U-U/10) \text{ OR } (A52 - A51) > (U+U/10) \\
& \text{CREATE A MESSAGE CALLED ERROR} \\
& \text{DISPLAY ERROR WITH "error5c.icn" AT } \langle 50, 275 \rangle \\
& \text{READ AS/} \\
& \text{ALWAYS} \\
\end{align*}

ALWAYS

\begin{align*}
& \text{IF}(A52 - A51) <= (U-U/10) \text{ AND } (A52 - A51) >= (U-U/10) \quad \text{"TIME WINDOW"} \\
& \text{LET } S5 = 0 \\
& \text{LET } A51 = \text{TIME.V} \\
& \text{LET } U = .M5.W2 \\
& \text{LET } X5 = \text{LOCATION.X(M5)} \\
& \text{LET } Y5 = \text{LOCATION.Y(M5)} \\
& \text{CREATE A STATE CALLED M5} \\
& \text{DISPLAY M5 WITH "wlu.icn" AT } \langle X5+U*.L, Y5+.H \rangle \\
& \text{CREATE A STATE CALLED M5} \\
& \text{DISPLAY M5 WITH "sl.icn" AT } \langle X5, Y5+.H \rangle \\
& \text{CREATE A STATE CALLED M5} \\
& \text{DISPLAY M5 WITH "su.icn" AT } \langle X5+U*.L, Y5+.H \rangle \\
& \text{WAIT } U \text{ UNITS} \\
& \text{LET } A52 = \text{TIME.V} \\
\end{align*}

ELSE

\begin{align*}
& \text{IF}(A52 - A51) < (U-U/10) \\
& \text{CREATE A MESSAGE CALLED ERROR} \\
& \text{DISPLAY ERROR WITH "error5a.icn" AT } \langle 50, 275 \rangle \\
& \text{READ AS/} \\
& \text{STOP.} \\
\end{align*}
ALWAYS
IF (A52-A51) > (U+U/10)
   CREATE A MESSAGE CALLED ERROR
   DISPLAY ERROR WITH "error5b.icn" AT (50, 275)
   READ AS /
   STOP
ALWAYS

ALWAYS
IF (A52-A51) <= (U+U/10) AND (A52-A51) >= (U-U/10) " TIME WINDOW
   LET S5 = 1
   LET A51 = TIME.V
   LET U = .M5.S
   LET X5 = LOCATION.X(M5)
   LET Y5 = LOCATION.Y(M5)
   CREATE A STATE CALLED M5
   DISPLAY M5 WITH "wld.icn" AT (X5+U*.L, Y5+.H)
   CREATE A STATE CALLED M5
   DISPLAY M5 WITH "sl.icn" AT (X5, Y5+.H)
   CREATE A STATE CALLED M5
   DISPLAY M5 WITH "sd.icn" AT (X5+U*.L, Y5+.H)
   WORK U UNITS
   LET A52 = TIME.V
ELSE
   IF (A52-A51) < (U-U/10) OR (A52-A51) > (U+U/10)
      CREATE A MESSAGE CALLED ERROR
      DISPLAY ERROR WITH "error5c.icn" AT (50, 275)
      READ AS /
      STOP
ALWAYS

ALWAYS
IF (A52-A51) <= (U+U/10) AND (A52-A51) >= (U-U/10) " TIME WINDOW
   LET S5 = 0
   LET A51 = TIME.V
   LET U = .M5.W
   LET X5 = LOCATION.X(M5)
   LET Y5 = LOCATION.Y(M5)
   CREATE A STATE CALLED M5
   DISPLAY M5 WITH "wld.icn" AT (X5+U*.L, Y5-.H)
   CREATE A STATE CALLED M5
   DISPLAY M5 WITH "s2.icn" AT (X5, Y5-.H)
   CREATE A STATE CALLED M5
   DISPLAY M5 WITH "sd.icn" AT (X5+U*.L, Y5-.H)
   WAIT U UNITS
   LET A52 = TIME.V
ELSE
   IF (A52-A51) < (U-U/10)
      CREATE A MESSAGE CALLED ERROR
      DISPLAY ERROR WITH "error5a.icn" AT (50, 275)
      READ AS /
      STOP
ALWAYS
IF (A52-A51) > (U+U/10)
   CREATE A MESSAGE CALLED ERROR
   DISPLAY ERROR WITH "error5b.icn" AT (50, 275)
   READ AS /
   STOP
ALWAYS

ALWAYS
IF (A52-A51) <= (U+U/10) AND (A52-A51) >= (U-U/10) " TIME WINDOW
   LET S5 = 1
   LET A51 = TIME.V
   LET U = .M5.S
   LET X5 = LOCATION.X(M5)
   LET Y5 = LOCATION.Y(M5)
   CREATE A STATE CALLED M5
   DISPLAY M5 WITH "wld.icn" AT (X5+U*.L, Y5-.H)
   CREATE A STATE CALLED M5
   DISPLAY M5 WITH "sl.icn" AT (X5, Y5-.H)
   CREATE A STATE CALLED M5
   DISPLAY M5 WITH "sd.icn" AT (X5+U*.L, Y5-.H)
DEFINE U, A61 AND A62 AS INTEGER VARIABLES
DEFINE M6 AS A POINTER VARIABLE
DEFINE X6, Y6 AS REAL VARIABLES
CREATE A MESSAGE CALLED LABEL
DISPLAY LABEL WITH "name6.icn" AT (30, Y-150+25)
LET S6 = 0
LET A61 = TIME.V
LET U = .M6.W1

**m6.sim**

PROCESS MM6
DEFINE U, A61 AND A62 AS INTEGER VARIABLES
DEFINE M6 AS A POINTER VARIABLE
DEFINE X6, Y6 AS REAL VARIABLES
CREATE A MESSAGE CALLED LABEL
DISPLAY LABEL WITH "name6.icn" AT (30, Y-150+25)
LET S6 = 0
LET A61 = TIME.V
LET U = .M6.W1
CREATE A STATE CALLED M6
DISPLAY M6 WITH "w2u.icn" AT (X+U*.L, Y-150)
CREATE A STATE CALLED M6
DISPLAY M6 WITH "s21.icn" AT (X, Y-150)
CREATE A STATE CALLED M6
DISPLAY M6 WITH "su.icn" AT (X+U*.L, Y-150)
WAIT U UNITS
LET A62 = TIME.V
IF (A62-A61) <= (U+U/10) AND (A62-A61) >= (U-U/10) " TIME WINDOW
   LET S6 = 1
   LET A61 = TIME.V
   LET U = .M6.S
   LET X6 = LOCATION.X(M6)
   LET Y6 = LOCATION.Y(M6)
   CREATE A STATE CALLED M6
   DISPLAY M6 WITH "w2d.icn" AT (X6+U*.L, Y6+.H)
   CREATE A STATE CALLED M6
   DISPLAY M6 WITH "s2.icn" AT (X6, Y6+.H)
   CREATE A STATE CALLED M6
   DISPLAY M6 WITH "sd.icn" AT (X6+U*.L, Y6+.H)
   WORK U UNITS
   LET A62 = TIME.V
ELSE
   IF (A62-A61) < (U-U/10) OR (A62-A61) > (U+U/10)
      CREATE A MESSAGE CALLED ERROR
      DISPLAY ERROR WITH "errorc6.icn" AT (50, 275)
      READ AS/
      STOP
   ALWAYS
   IF (A62-A61) <= (U+U/10) AND (A62-A61) >= (U-U/10) " TIME WINDOW
   LET S6 = 0
   LET A61 = TIME.V
   LET U = .M6.W2
   LET X6 = LOCATION.X(M6)
   LET Y6 = LOCATION.Y(M6)
   CREATE A STATE CALLED M6
   DISPLAY M6 WITH "w2u.icn" AT (X6+U*.L, Y6-.H)
   CREATE A STATE CALLED M6
   DISPLAY M6 WITH "s4.icn" AT (X6, Y6-.H)
   CREATE A STATE CALLED M6
   DISPLAY M6 WITH "su.icn" AT (X6+U*.L, Y6-.H)
   WAIT U UNITS
   LET A62 = TIME.V
ELSE
   IF (A62-A61) LT (U-U/10)
      CREATE A MESSAGE CALLED ERROR
      DISPLAY ERROR WITH "error6a.icn" AT (50, 275)
      READ AS/
      STOP
   ALWAYS
   IF (A62-A61) GT (U+U/10)
      CREATE A MESSAGE CALLED ERROR
      DISPLAY ERROR WITH "error6b.icn" AT (50, 275)
      READ AS/
      STOP
   ALWAYS
ALWAYS
IF (A62-A61) <= (U+U/10) AND (A62-A61) >= (U-U/10) " TIME WINDOW
   LET S6 = 1
   LET A61 = TIME.V
   LET U = .M6.S
   LET X6 = LOCATION.X(M6)
   LET Y6 = LOCATION.Y(M6)
   CREATE A STATE CALLED M6
   DISPLAY M6 WITH "w2d.icn" AT (X6+U*.L, Y6+.H)
   CREATE A STATE CALLED M6
   DISPLAY M6 WITH "s2.icn" AT (X6, Y6+.H)
   CREATE A STATE CALLED M6
   DISPLAY M6 WITH "sd.icn" AT (X6+U*.L, Y6+.H)
   CREATE A STATE CALLED M6
DISPLAY M6 WITH "sd.icn" AT (X6+U*L, Y6+H)
WORK U UNITS
LET A62 = TIME.V

ELSE
IF(A62 - A61) < (U-U/10) OR (A62 - A61) > (U+U/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "error6a.icn" AT (50, 275)
READ AS /
STOP

ALWAYS

ALWAYS
IF (A62 - A61) <= (U+U/10) AND (A62 - A61) >= (U-U/10) " TIME WINDOW
LET S6 = 0
LET A61 = TIME.V
LET U = .M6.W3
LET X6 = LOCATION.X(M6)
LET Y6 = LOCATION.Y(M6)
CREATE A STATE CALLED M6
DISPLAY M6 WITH "w2u.icn" AT (X6+U*L, Y6-H)
CREATE A STATE CALLED M6
DISPLAY M6 WITH "s8.icn" AT (X6, Y6-H)
CREATE A STATE CALLED M6
DISPLAY M6 WITH "su.icn" AT (X6+U*L, Y6-H)
WAIT U UNITS
LET A62 = TIME.V

ELSE
IF(A62 - A61) LT (U-U/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "error6a.icn" AT (50, 275)
READ AS /
STOP
ALWAYS
IF (A62 - A61) GT (U+U/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "error6b.icn" AT (50, 275)
READ AS /
STOP

ALWAYS

ALWAYS
IF (A62 - A61) <= (U+U/10) AND (A62 - A61) >= (U-U/10) " TIME WINDOW
LET S6 = 1
LET A61 = TIME.V
LET U = .M6.W3
LET X6 = LOCATION.X(M6)
LET Y6 = LOCATION.Y(M6)
CREATE A STATE CALLED M6
DISPLAY M6 WITH "w2d.icn" AT (X6+U*L, Y6+H)
CREATE A STATE CALLED M6
DISPLAY M6 WITH "s2.icn" AT (X6, Y6+H)
CREATE A STATE CALLED M6
DISPLAY M6 WITH "sd.icn" AT (X6+U*L, Y6+H)
WORK U UNITS
LET A62 = TIME.V

ELSE
IF(A62 - A61) < (U-U/10) OR (A62 - A61) > (U+U/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "error6a.icn" AT (50, 275)
READ AS /
STOP

ALWAYS

ALWAYS
IF (A62 - A61) <= (U+U/10) AND (A62 - A61) >= (U-U/10) " TIME WINDOW
LET S6 = 0
LET A61 = TIME.V
LET U = .M6.W3
LET X6 = LOCATION.X(M6)
LET Y6 = LOCATION.Y(M6)
CREATE A STATE CALLED M6
DISPLAY M6 WITH "w2u.icn" AT (X6+U*L, Y6-H)
CREATE A STATE CALLED M6
DISPLAY M6 WITH "s8.icn" AT (X6, Y6-.H)
CREATE A STATE CALLED M6
DISPLAY M6 WITH "su.icn" AT (X6+U*.L, Y6-.H)
WAIT U UNITS
LET A62 = TIME.V
ELSE
   IF(A62-A61) LT (U-U/10)
      CREATE A MESSAGE CALLED ERROR
      DISPLAY ERROR WITH "error6a.icn" AT (50,275)
      READ AS/
      STOP
   ALWAYS
   IF (A62-A61) GT (U+U/10)
      CREATE A MESSAGE CALLED ERROR
      DISPLAY ERROR WITH "error6b.icn" AT (50,275)
      READ AS/
      STOP
   ALWAYS
   ALWAYS
   IF (A62-A61) <= (U+U/10) AND (A62-A61) >= (U-U/10) " TIME WINDOW
      LET S6 = 1
      LET A61 = TIME.V
      LET U = .M6.S
      LET X6 = LOCATION.X(M6)
      LET Y6 = LOCATION.Y(M6)
      CREATE A STATE CALLED M6
      DISPLAY M6 WITH "w2d.icn" AT (X6+U*.L, Y6+.H)
      CREATE A STATE CALLED M6
      DISPLAY M6 WITH "s2.icn" AT (X6, Y6+.H)
      CREATE A STATE CALLED M6
      DISPLAY M6 WITH "sd.icn" AT (X6+U*.L, Y6+.H)
      WORK U UNITS
      LET A62 = TIME.V
      ELSE
         IF(A62-A61) < (U-U/10) OR (A62-A61) > (U+U/10)
            CREATE A MESSAGE CALLED ERROR
            DISPLAY ERROR WITH "error6a.icn" AT (50,275)
            READ AS/
            STOP
         ALWAYS
         ALWAYS
         IF (A62-A61) <= (U+U/10) AND (A62-A61) >= (U-U/10) " TIME WINDOW
            LET S6 = 0
            LET A61 = TIME.V
            LET U = .M6.W4
            LET X6 = LOCATION.X(M6)
            LET Y6 = LOCATION.Y(M6)
            CREATE A STATE CALLED M6
            DISPLAY M6 WITH "w2u.icn" AT (X6+U*.L, Y6-.H)
            CREATE A STATE CALLED M6
            DISPLAY M6 WITH "sl0.icn" AT (X6, Y6-.H)
            CREATE A STATE CALLED M6
            DISPLAY M6 WITH "su.icn" AT (X6+U*.L, Y6-.H)
            WAIT U UNITS
            LET A62 = TIME.V
         ELSE
            IF(A62-A61) LT (U-U/10)
               CREATE A MESSAGE CALLED ERROR
               DISPLAY ERROR WITH "error6a.icn" AT (50, 275)
               READ AS/
               STOP
            ALWAYS
            IF (A62-A61) GT (U+U/10)
               CREATE A MESSAGE CALLED ERROR
               DISPLAY ERROR WITH "error6b.icn" AT (50, 275)
               READ AS/
               STOP
            ALWAYS
Appendix D

ALWAYS
IF (A62-A61) <= (U+U/10) AND (A62-A61) >= (U-U/10) " TIME WINDOW
LET S6 = 1
LET A61 = TIME.V
LET U = .M6.S
LET X6 = LOCATION.X(M6)
LET Y6 = LOCATION.Y(M6)
CREATE A STATE CALLED M6
DISPLAY M6 WITH "w2d.icn" AT (X6+U*.L, Y6+.H)
CREATE A STATE CALLED M6
DISPLAY M6 WITH "s2.icn" AT (X6, Y6+.H)
CREATE A STATE CALLED M6
DISPLAY M6 WITH "sd.icn" AT (X6+U*.L, Y6+.H)
ORK U UNITS
LET A62 = TIME.V
ELSE
IF (A62-A61) < (U-U/10) OR (A62-A61) > (U+U/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "error6a.icn" AT (50, 275)
READ AS /
STOP
ALWAYS
ALWAYS
IF (A62-A61) <= (U+U/10) AND (A62-A61) >= (U-U/10) " TIME WINDOW
LET S6 = 0
LET A61 = TIME.V
LET U = .M6.W3
LET X6 = LOCATION.X(M6)
LET Y6 = LOCATION.Y(M6)
CREATE A STATE CALLED M6
DISPLAY M6 WITH "w2u.icn" AT (X6+U*.L, Y6-.H)
CREATE A STATE CALLED M6
DISPLAY M6 WITH "s8.icn" AT (X6, Y6-.H)
CREATE A STATE CALLED M6
DISPLAY M6 WITH "su.icn" AT (X6+U*.L, Y6-.H)
WAIT U UNITS
LET A62 = TIME.V
ELSE
IF(A62-A61) LT (U-U/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "error6a.icn" AT (50, 275)
READ AS /
STOP
ALWAYS
IF (A62-A61) GT (U+U/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "error6b.icn" AT (50, 275)
READ AS /
STOP
ALWAYS
ALWAYS
IF (A62-A61) <= (U+U/10) AND (A62-A61) >= (U-U/10) " TIME WINDOW
LET S6 = 1
LET A61 = TIME.V
LET U = .M6.S
LET X6 = LOCATION X(M6)
LET Y6 = LOCATION Y(M6)
CREATE A STATE CALLED M6
DISPLAY M6 WITH "w2d.icn" AT (X6+U*.L, Y6+.H)
CREATE A STATE CALLED M6
DISPLAY M6 WITH "s2.icn" AT (X6, Y6+.H)
CREATE A STATE CALLED M6
DISPLAY M6 WITH "sd.icn" AT (X6+U*.L, Y6+.H)
ORK U UNITS
LET A62 = TIME.V
ELSE
IF(A62-A61) < (U-U/10) OR (A62-A61) > (U+U/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "error6a.icn" AT (50,275) 
READ AS /
STOP 
ALWAYS

ALWAYS
IF ((A62-A61) <= (U+U/10) AND (A62-A61) >= (U-U/10)) " TIME WINDOW
LET S6 = 0 
LET A61 = TIME.V 
LET U = .M6.W5 
LET X6 = LOCATION X(M6) 
CREATE A STATE CALLED M6 
DISPLAY M6 WITH "s5.icn" AT (X6, Y6-.H) 
WAIT U UNITS 
LET A62 = TIME.V 
ELSE 
IF (A62-A61) LT (U-U/10)
CREATE A MESSAGE CALLED ERROR 
DISPLAY ERROR WITH "error6a.icn" AT (50,275) 
READ AS /
STOP 
ALWAYS
IF (A62-A61) GT (U+U/10)
CREATE A MESSAGE CALLED ERROR 
DISPLAY ERROR WITH "error6b.icn" AT (50,275) 
READ AS /
STOP 
ALWAYS
ALWAYS
IF (A62-A61) <= (U+U/10) AND (A62-A61) >= (U-U/10) " TIME WINDOW
LET S6 = 0 
ELSE 
IF (A62-A61) < (U-U/10) OR (A62-A61) > (U+U/10)
CREATE A MESSAGE CALLED ERROR 
DISPLAY ERROR WITH "error6c.icn" AT (50,275) 
READ AS /
STOP 
ALWAYS
ALWAYS
END " END OF M6

---

**cm21.sim**

PROCESS CM21
DEFINE MM AS A POINTER VARIABLE
CREATE A MESSAGE CALLED LABEL
DISPLAY LABEL WITH "namec21.icn" AT (30, Y-300+25) 
DEFINE XM, YM AS REAL VARIABLES 
LET XM = X 
LET YM = Y-300.0
'LABEL1' 
WHILE TIME.V LT .CYCLE.TIME 
DO
UNTIL S5 = 1 OR S6 = 1
DO
WAIT 1 UNIT 
CREATE A STATE CALLED MM 
DISPLAY MM WITH "s1.icn" AT (XM, YM) 
LET XM = LOCATION X(MM)+.L 
LET YM = LOCATION Y(MM) 
IF TIME.V >= CYCLE.TIME 
GO TO LABEL1
ALWAYS
LOOP 
CREATE A STATE CALLED MM 
DISPLAY MM WITH "w2u.icn" AT (XM, YM) 
CREATE A STATE CALLED MM 
DISPLAY MM WITH "su.icn" AT (XM, YM) 
LET YM = LOCATION Y(MM)+.H
UNTIL $S5 = 0$ AND $S6 = 0$
DO
    WAIT 1 UNIT
    CREATE A STATE CALLED MM
    DISPLAY MM WITH "sl.icn" AT (XM, YM)
    LET XM = LOCATION.X(MM)+.L
    LET YM = LOCATION.Y(MM)
LOOP
CREATE A STATE CALLED MM
DISPLAY MM WITH "w2d.icn" AT (XM, YM)
CREATE A STATE CALLED MM
DISPLAY MM WITH "sd.icn" AT (XM, YM)
LET YM = LOCATION.Y(MM)-.H
LOOP
END " END OF CM21

D.5 Code for Control Signal of CM3

preamble.sim

PREAMBLE
PROCESSES INCLUDE MM7, MM8, CM3
DEFINE S7 AND S8 AS INTEGER VARIABLES
DEFINE ERROR, LABEL, REPORT AS POINTER VARIABLES
DEFINE X, Y AS REAL VARIABLES
GRAPHIC ENTITIES INCLUDE STATE AND MESSAGE
DEFINE .L TO MEAN 25 0
DEFINE .H TO MEAN 50.0
DEFINE .M7.S TO MEAN 4
DEFINE .M7.W1 TO MEAN 4
DEFINE .M7.W2 TO MEAN 8
DEFINE .M7.W3 TO MEAN 18
DEFINE .M8.S TO MEAN 2
DEFINE .M8.W1 TO MEAN 14
DEFINE .M8.W2 TO MEAN 4
DEFINE .M8.W3 TO MEAN 16
DEFINE CYCLE.TIME TO MEAN 38
END " END OF PREAMBLE

main.sim

MAIN
LET X = 30
LET Y = 700
LET VXFORM.V = 1
LET TIMESCALE.V = 100
CALL SETWORLD.R(0.0, 1000.0, 0.0, 1000.0)
CREATE A MESSAGE CALLED REPORT
DISPLAY REPORT WITH "title.icn" AT (501,900)
CREATE A MESSAGE CALLED REPORT
DISPLAY REPORT WITH "heading.icn" AT (501,850)
ACTIVATE A MM7 NOW
ACTIVATE A MM8 NOW
ACTIVATE A CM3 NOW
START SIMULATION
CREATE A MESSAGE CALLED REPORT
DISPLAY REPORT WITH "legend.icn" AT (501,850)
READ AS /
END " END OF MAIN
**cm7.sim**

PROCESS MM7
DEFINE U, A71 AND A72 AS INTEGER VARIABLES
DEFINE M7 AS A POINTER VARIABLE
DEFINE X7, Y7 AS REAL VARIABLES
CREATE A MESSAGE CALLED LABEL
DISPLAY LABEL WITH "name7.icn" AT (30, Y+25)
LET S7 = 0
LET A71 = TIME.V
LET U = M7.W1
CREATE A STATE CALLED M7
DISPLAY M7 WITH "w2u.icn" AT (X+U*.L, Y)
CREATE A STATE CALLED M7
DISPLAY M7 WITH "s4.icn" AT (X, Y)
CREATE A STATE CALLED M7
DISPLAY M7 WITH "su.icn" AT (X+U*.L, Y)
WAIT U UNITS
LET A72 = TIME.V
IF (A72-A71) <= (U+U/10) AND (A72-A71) >= (U-U/10) " TIME WINDOW
LET S7 = 1
LET A71 = TIME.V
LET U = M7.S
LET X7 = LOCATION.X(M7)
LET Y7 = LOCATION.Y(M7)
CREATE A STATE CALLED M7
DISPLAY M7 WITH "w1d.icn" AT (X7+U*.L, Y7+.H)
CREATE A STATE CALLED M7
DISPLAY M7 WITH "s4.icn" AT (X7, Y7+.H)
CREATE A STATE CALLED M7
DISPLAY M7 WITH "sd.icn" AT (X7+U*.L, Y7+.H)
WORK U UNITS
LET A72 = TIME.V
ELSE
IF(A72-A71) < (U-U/10) OR (A72-A71) > (U+U/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "error7.icn" AT (50,275)
READ AS /
STOP
ALWAYS
IF (A72-A71) <= (U+U/10) AND (A72-A71) >= (U-U/10) " TIME WINDOW
LET S7 = 0
LET A71 = TIME.V
LET U = M7.W2
LET X7 = LOCATION.X(M7)
LET Y7 = LOCATION.Y(M7)
CREATE A STATE CALLED M7
DISPLAY M7 WITH "w2d.icn" AT (X7+U*.L, Y7-.H)
CREATE A STATE CALLED M7
DISPLAY M7 WITH "s8.icn" AT (X7, Y7-.H)
CREATE A STATE CALLED M7
DISPLAY M7 WITH "sd.icn" AT (X7+U*.L, Y7-.H)
WORK U UNITS
LET A72 = TIME.V
ELSE
IF(A72-A71) < (U-U/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "error7a.icn" AT (50, 275)
READ AS /
STOP
ALWAYS
IF(A72-A71) > (U+U/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "error7b.icn" AT (50, 275)
READ AS /
STOP
ALWAYS

IF (A72-A71) <= (U+U/10) AND (A72-A71) >= (U-U/10) " TIME WINDOW
LET S7 = 1
LET A71 = TIME.V
LET U = .M7.S
LET X7 = LOCATION.X(M7)
LET Y7 = LOCATION.Y(M7)
CREATE A STATE CALLED M7
DISPLAY M7 WITH "w1u.icn" AT (X7+U*.L, Y7-.H)
CREATE A STATE CALLED M7
DISPLAY M7 WITH "s4.icn" AT (X7, Y7-.H)
CREATE A STATE CALLED M7
DISPLAY M7 WITH "su.icn" AT (X7+U*.L, Y7-.H)
WORK U UNITS
LET A72 = TIME.V

ELSE
IF(A72-A71) < (U-U/10) OR (A72-A71) > (U+U/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "error7.icn" AT (50, 275)
READ AS/
ALWAYS

ALWAYS
IF (A72-A71) <= (U+U/10) AND (A72-A71) >= (U-U/10) " TIME WINDOW
LET S7 = 0
LET A71 = TIME.V
LET U = .M7.W3
CREATE A STATE CALLED M7
DISPLAY M7 WITH "s18.icn" AT (X7, Y7+.H)
WAIT .M7.W3 UNITS

ELSE
IF (A72-A71) < (U-U/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "error7a.icn" AT (50, 275)
READ AS/
STOP
ALWAYS
IF(A72-A71) > (U+U/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "error7b.icn" AT (50, 275)
READ AS/
STOP
ALWAYS

ALWAYS

END " END OF MM7

m8.sim

PROCESS MM8
DEFINE U, A81 AND A82 AS INTEGER VARIABLES
DEFINE M8 AS A POINTER VARIABLE
DEFINE X8, Y8 AS REAL VARIABLES
CREATE A MESSAGE CALLED LABEL
DISPLAY LABEL WITH "name8.icn" AT (30, Y-150+25)
LET S8 = 0
LET A81 = TIME.V
LET U = .M8.W1
CREATE A STATE CALLED M8
DISPLAY M8 WITH "w2u.icn" AT (X+U*.L, Y-150)
CREATE A STATE CALLED M8
DISPLAY M8 WITH "s4.icn" AT (X, Y-150)
CREATE A STATE CALLED M8
DISPLAY M8 WITH "su.icn" AT (X+U*.L, Y-150)
WAIT U UNITS
LET A82 = TIME.V
IF (A82-A81) <= (U+U/10) AND (A82-A81) >= (U-U/10) " TIME WINDOW
LET S8 = 1
LET A81 = TIME.V
LET U = .M8.S
LET X8 = LOCATION.X(M8)
LET Y8 = LOCATION.Y(M8)
CREATE A STATE CALLED M8
DISPLAY M8 WITH "wld.icn" AT (X8+U*.L, Y8+.H)
CREATE A STATE CALLED M8
DISPLAY M8 WITH "s2.icn" AT (X8, Y8+.H)
CREATE A STATE CALLED M8
DISPLAY M8 WITH "sd.icn" AT (X8+U*.L, Y8+.H)
WORK U UNITS
LET A8 = TIME.V

ELSE

IF(A82-A81) < (U-U/10) OR (A82-A81) > (U+U/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "error8.icn" AT (50,275)
READ AS /
STOP

ALWAYS
IF (A82-A81) <= (U+U/10) AND (A82-A81) >= (U-U/10) " TIME WINDOW
LET S8 = 0
LET A81 = TIME.V
LET U = .M8.W2
LET X8 = LOCATION.X(M8)
LET Y8 = LOCATION.Y(M8)
CREATE A STATE CALLED M8
DISPLAY M8 WITH "wld.icn" AT (X8+U*.L, Y8-.H)
CREATE A STATE CALLED M8
DISPLAY M8 WITH "s4.icn" AT (X8, Y8-.H)
CREATE A STATE CALLED M8
DISPLAY M8 WITH "sd.icn" AT (X8+U*.L, Y8-.H)
WAIT U UNITS
LET A82 = TIME.V

ELSE

IF(A82-A81) LT (U-U/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "error8a.icn" AT (50,275)
READ AS /
STOP

ALWAYS
IF (A82-A81) GT (U+U/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "error8b.icn" AT (50,275)
READ AS /
STOP

ALWAYS
IF (A82-A81) <= (U+U/10) AND (A82-A81) >= (U-U/10) " TIME WINDOW
LET S8 = 1
LET A81 = TIME.V
LET U = .M8.S
LET X8 = LOCATION.X(M8)
LET Y8 = LOCATION.Y(M8)
CREATE A STATE CALLED M8
DISPLAY M8 WITH "wlu.icn" AT (X8+U*.L, Y8-.H)
CREATE A STATE CALLED M8
DISPLAY M8 WITH "s2.icn" AT (X8, Y8-.H)
CREATE A STATE CALLED M8
DISPLAY M8 WITH "su.icn" AT (X8+U*.L, Y8-.H)
WORK U UNITS
LET A82 = TIME.V

ELSE

IF(A82-A81) < (U-U/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "error8a.icn" AT (50,275)
READ AS /
STOP

ALWAYS
IF (A82-A81) > (U+U/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "error8b.icn" AT (50, 275)
READ AS /
STOP
ALWAYS

IF (A82-A81) <= (U-U/10) AND (A82-A81) >= (U-U/10) "TIME WINDOW"
LET S8 = 0
LET X8 = LOCATION.X(M8)
LET Y8 = LOCATION.Y(M8)
CREATE A STATE CALLED M8
DISPLAY M8 WITH "sl6.icn" AT (X8, Y8+.H)
WAIT .M8.W3 UNITS
ELSE
IF (A82-A81) < (U-U/10) OR (A82-A81) > (U+U/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "errorc8.icn" AT (50, 275)
READ AS /
STOP
ALWAYS
END
"END OF M8

**cm3.sim**

PROCESS CM3
DEFINE MM AS A POINTER VARIABLE
CREATE A MESSAGE CALLED LABEL
DISPLAY LABEL WITH "namec3.icn" AT (30, Y-300+25)
DEFINE XM, YM AS REAL VARIABLES
LET XM = X
LET YM = Y-300.0
"LABEL1" WHILE TIME.V LT .CYCLE.TIME
DO
UNTIL S7 = 1 OR S8 = 1
DO
WAIT 1 UNIT
CREATE A STATE CALLED MM
DISPLAY MM WITH "sl.icn" AT (XM, YM)
LET XM = LOCATION.X(MM)+.L
LET YM = LOCATION.Y(MM)
IF TIME.V >= .CYCLE.TIME
GO TO LABEL1
ALWAYS
LOOP
CREATE A STATE CALLED MM
DISPLAY MM WITH "w2u.icn" AT (XM, YM)
CREATE A STATE CALLED MM
DISPLAY MM WITH "su.icn" AT (XM, YM)
LET YM = LOCATION.Y(MM)+.H
UNTIL S7 = 0 AND S8 = 0
DO
WAIT 1 UNIT
CREATE A STATE CALLED MM
DISPLAY MM WITH "sl.icn" AT (XM, YM)
LET XM = LOCATION.X(MM)+.L
LET YM = LOCATION.Y(MM)
LOOP
CREATE A STATE CALLED MM
DISPLAY MM WITH "w2d.icn" AT (XM, YM)
CREATE A STATE CALLED MM
DISPLAY MM WITH "sd.icn" AT (XM, YM)
LET YM = LOCATION.Y(MM)-.H
LOOP
END
" END OF CM3
D.6 Code for Control Signal of CM4

**preamble.sim**

PREAMBLE

PROCESSES INCLUDE MM9, MM10, CM4

DEFINE S9 AND S10 AS INTEGER VARIABLES

DEFINE ERROR, LABEL, REPORT AS POINTER VARIABLES

DEFINE X, Y AS REAL VARIABLES

GRAPHIC ENTITIES INCLUDE STATE AND MESSAGE

DEFINE .L TO MEAN 50.0

DEFINE .I TO MEAN 50.0

DEFINE .M9.W1 TO MEAN 13

DEFINE .M9.W2 TO MEAN 4

DEFINE .M9.S TO MEAN 1

DEFINE .M10.S TO MEAN 1

DEFINE .M10.W1 TO MEAN 14

DEFINE .M10.W2 TO MEAN 2

DEFINE .CYCLE.TIME TO MEAN 19

END " END OF PREAMBLE

**main.sim**

MAIN

LET X = 30

LET Y = 700

LET VXFORM.V = 1

LET TIMESCALE.V = 100

CALL SETWORLD.R(0.0, 1000.0, 0.0, 1000.0)

CREATE A MESSAGE CALLED REPORT

DISPLAY REPORT WITH "title.icn" AT (501,900)

CREATE A MESSAGE CALLED REPORT

DISPLAY REPORT WITH "heading.icn" AT (501,850)

ACTIVATE A MM9 NOW

ACTIVATE A MM10 NOW

ACTIVATE A CM4 NOW

START SIMULATION

CREATE A MESSAGE CALLED REPORT

DISPLAY REPORT WITH "legend.icn" AT (50,275)

READ AS/

END " END OF MAIN

**m9.sim**

PROCESS MM9

DEFINE U, M91 AND M92 AS INTEGER VARIABLES

DEFINE M9 AS A POINTER VARIABLE

DEFINE X9, Y9 AS REAL VARIABLES

CREATE A MESSAGE CALLED LABEL

DISPLAY LABEL WITH "name9.icn" AT (50, Y+25)

LET S9 = 0

LET M91 = TIME.V

LET U = .M9.W1

CREATE A STATE CALLED M9

DISPLAY M9 WITH "w2u.icn" AT (X+U*.L, Y)

CREATE A STATE CALLED M9

DISPLAY M9 WITH "sl3.icn" AT (X, Y)

CREATE A STATE CALLED M9

DISPLAY M9 WITH "su.icn" AT (X+U*.L, Y)
WAIT U UNITS
LET M92 = TIME.V
IF (M92-M91) <= (U+U/10) AND (M92-M91) >= (U-U/10) * TIME WINDOW
    LET S9 = 1
    LET M91 = TIME.V
    LET U = .M9.S
    LET X9 = LOCATION.X(M9)
    LET Y9 = LOCATION.Y(M9)
    CREATE A STATE CALLED M9
    DISPLAY M9 WITH "wld.icn" AT (X9+U*.L, Y9+.H)
    CREATE A STATE CALLED M9
    DISPLAY M9 WITH "sl.icn" AT (X9, Y9+.H)
    CREATE A STATE CALLED M9
    DISPLAY M9 WITH "sd.icn" AT (X9+U*.L, Y9+.H)
    WORK U UNITS
    LET M92 = TIME.V
ELSE
    IF(M92-M91) < (U-U/10) OR (M92-M91) > (U+U/10)
        CREATE A MESSAGE CALLED ERROR
        DISPLAY ERROR WITH "error9.icn" AT (50,275)
        READ AS /
        STOP
    ALWAYS
ELSE
    IF(M92-M91) <= (U-U/10) AND (M92-M91) >= (U-U/10) * TIME WINDOW
    LET S9 = 0
    LET M91 = TIME.V
    LET U = .M9.W2
    LET X9 = LOCATION.X(M9)
    LET Y9 = LOCATION.Y(M9)
    CREATE A STATE CALLED M9
    DISPLAY M9 WITH "w2d.icn" AT (X9+U*.L, Y9-.H)
    CREATE A STATE CALLED M9
    DISPLAY M9 WITH "s4.icn" AT (X9, Y9-.H)
    CREATE A STATE CALLED M9
    DISPLAY M9 WITH "sd.icn" AT (X9+U*.L, Y9-.H)
    WAIT U UNITS
    LET M92 = TIME.V
ELSE
    IF(M92-M91) < (U-U/10)
        CREATE A MESSAGE CALLED ERROR
        DISPLAY ERROR WITH "error9a.icn" AT (50,275)
        READ AS /
        STOP
    ALWAYS
    IF(M92-M91) > (U+U/10)
        CREATE A MESSAGE CALLED ERROR
        DISPLAY ERROR WITH "error9b.icn" AT (50,275)
        READ AS /
        STOP
    ALWAYS
ELSE
    IF (M92-M91) <= (U+U/10) AND (M92-M91) >= (U-U/10) * TIME WINDOW
    LET S9 = 1
    LET M91 = TIME.V
    LET U = .M9.S
    LET X9 = LOCATION.X(M9)
    LET Y9 = LOCATION.Y(M9)
    CREATE A STATE CALLED M9
    DISPLAY M9 WITH "wlu.icn" AT (X9+U*.L, Y9-.H)
    CREATE A STATE CALLED M9
    DISPLAY M9 WITH "sl.icn" AT (X9, Y9-.H)
    CREATE A STATE CALLED M9
    DISPLAY M9 WITH "su.icn" AT (X9+U*.L, Y9-.H)
    WORK U UNITS
    LET M92 = TIME.V
ELSE
    IF(M92-M91) < (U-U/10) OR (M92-M91) > (U+U/10)
        CREATE A MESSAGE CALLED ERROR
        DISPLAY ERROR WITH "error9a.icn" AT (50,275)
        READ AS /
        STOP
    ALWAYS
    IF(M92-M91) > (U+U/10)
        CREATE A MESSAGE CALLED ERROR
        DISPLAY ERROR WITH "error9b.icn" AT (50,275)
        READ AS /
        STOP
    ALWAYS

DISPLAY ERROR WITH "error9.icn" AT (50, 275)
READ AS /
ALWAYS
IF (M92-M91) <= (U+U/10) AND (M92-M91) >= (U-U/10) " TIME WINDOW
LET S9 = 0
ELSE
IF (M92-M91) < (U-U/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "error9a.icn" AT (50, 275)
READ AS /
STOP
ALWAYS
IF (M92-M91) > (U+U/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "error9b.icn" AT (50, 275)
READ AS /
STOP
ALWAYS
ALWAYS
END " END OF M9

m10.sim

PROCESS MM10
DEFINE U, A101 AND A102 AS INTEGER VARIABLES
DEFINE M10 AS A POINTER VARIABLE
DEFINE X10, Y10 AS REAL VARIABLES
CREATE A MESSAGE CALLED LABEL
DISPLAY LABEL WITH "name10.icn" AT (50, Y-150+25)
LET S10 = 0
LET A101 = TIME.V
LET U = .M10.W1
CREATE A STATE CALLED M10
DISPLAY M10 WITH "w2u.icn" AT (X+U*.L, Y-150)
CREATE A STATE CALLED M10
DISPLAY M10 WITH "sl4.icn" AT (X, Y-150)
CREATE A STATE CALLED M10
DISPLAY M10 WITH "su.icn" AT (X+U*.L, Y-150)
WAIT U UNITS
LET A10 = TIME.V
IF (A102-A101) <= (U-U/10) AND (A102-A101) >= (U+U/10) " TIME WINDOW
LET S10 = 1
LET A101 = TIME.V
LET U = .M10.S
LET X10 = LOCATION.X(M10)
LET Y10 = LOCATION.Y(M10)
CREATE A STATE CALLED M10
DISPLAY M10 WITH "w1d.icn" AT (X10+U*.L, Y10+.H)
CREATE A STATE CALLED M10
DISPLAY M10 WITH "s1.icn" AT (X10, Y10+.H)
CREATE A STATE CALLED M10
DISPLAY M10 WITH "sd.icn" AT (X10+U*.L, Y10+.H)
WORK U UNITS
LET A102 = TIME.V
ELSE
IF (A102-A101) < (U-U/10) OR (A102-A101) > (U+U/10)
CREATE A MESSAGE CALLED ERROR
DISPLAY ERROR WITH "error10.icn" AT (50, 275)
READ AS /
STOP
ALWAYS
ALWAYS
IF (A102-A101) <= (U+U/10) AND (A102-A101) >= (U-U/10) " TIME WINDOW
LET S10 = 0
LET A101 = TIME.V
LET U = .M10.W2
LET X10 = LOCATION.X(M10)
LET Y10 = LOCATION.Y(M10)
CREATE A STATE CALLED M10
DISPLAY M10 WITH "wlu.icn" AT (X10+L.U, Y10-H)
CREATE A STATE CALLED M10
DISPLAY M10 WITH "s2.icn" AT (X10, Y10-H)
CREATE A STATE CALLED M10
DISPLAY M10 WITH "su.icn" AT (X10+L.U, Y10-H)
WAIT U UNITS
LET A102 = TIME.V
ELSE
IF (A102-A101) LT (U-U/10)
    CREATE A MESSAGE CALLED ERROR
    DISPLAY ERROR WITH "error10a.icn" AT (50, 275)
    READ AS/
STOP
ALWAYS
IF (A102-A101) GT (U+U/10)
    CREATE A MESSAGE CALLED ERROR
    DISPLAY ERROR WITH "error10b.icn" AT (50, 275)
    READ AS/
STOP
ALWAYS
ALWAYS
IF (A102-A101) <= (U+U/10) AND (A102-A101) >= (U-U/10) " TIME WINDOW
LET S10 = 1
LET A101 = TIME.V
LET U = .M10.S
LET X10 = LOCATION.X(M10)
LET Y10 = LOCATION.Y(M10)
CREATE A STATE CALLED M10
DISPLAY M10 WITH "wld.icn" AT (X10+L.U, Y10+H)
CREATE A STATE CALLED M10
DISPLAY M10 WITH "sl.icn" AT (X10, Y10+H)
CREATE A STATE CALLED M10
DISPLAY M10 WITH "sd.icn" AT (X10+L.U, Y10+H)
WORK U UNITS
LET A102 = TIME.V
ELSE
IF (A102-A101) < (U-U/10)
    CREATE A MESSAGE CALLED ERROR
    DISPLAY ERROR WITH "error10a.icn" AT (50, 275)
    READ AS/
STOP
ALWAYS
IF (A102-A101) > (U+U/10)
    CREATE A MESSAGE CALLED ERROR
    DISPLAY ERROR WITH "error10b.icn" AT (50, 275)
    READ AS/
STOP
ALWAYS
ALWAYS
IF (A102-A101) <= (U+U/10) AND (A102-A101) >= (U-U/10) " TIME WINDOW
LET S10 = 0
LET X10 = LOCATION.X(M10)
LET Y10 = LOCATION.Y(M10)
CREATE A STATE CALLED M10
DISPLAY M10 WITH "s1.icn" AT (X10, Y10-H)
ELSE
IF (A102-A101) < (U-U/10) OR (A102-A101) > (U+U/10)
    CREATE A MESSAGE CALLED ERROR
    DISPLAY ERROR WITH "error10c.icn" AT (50, 275)
    READ AS/
STOP
ALWAYS
END " END OF M10
cm4.sim

PROCESS CM4
DEFINE MM AS A POINTER VARIABLE
CREATE A MESSAGE CALLED LABEL
DISPLAY LABEL WITH "namec4.icn" AT (50, Y-300+25)
DEFINE XM, YM AS REAL VARIABLES
LET XM = X
LET YM = Y-300.0
WHILE TIME.V LT CYCLE.TIME
DO
UNTIL S9 = 1 OR S10 = 1
DO
WAIT 1 UNIT
CREATE A STATE CALLED MM
DISPLAY MM WITH "s1.icn" AT (XM, YM)
LET XM = LOCATION.X(MM)+T
LET YM = LOCATION.Y(MM)
LOOP
CREATE A STATE CALLED MM
DISPLAY MM WITH "w2u.icn" AT (XM, YM)
CREATE A STATE CALLED MM
DISPLAY MM WITH "su.icn" AT (XM, YM)
LET YM = LOCATION.Y(MM)+H
UNTIL S9 = 0 AND S10 = 0
DO
WAIT 1 UNIT
CREATE A STATE CALLED MM
DISPLAY MM WITH "s1.icn" AT (XM, YM)
LET XM = LOCATION.X(MM)+L
LET YM = LOCATION.Y(MM)
LOOP
CREATE A STATE CALLED MM
DISPLAY MM WITH "w2d.icn" AT (XM, YM)
CREATE A STATE CALLED MM
DISPLAY MM WITH "sd.icn" AT (XM, YM)
LET YM = LOCATION.Y(MM)-H
LOOP
END
" END OF CM4
APPENDIX E

Genetic Algorithm Based Job Optimization Code

```c
#include<iostream.h>
#include<cstdio.h>
#include<stdlib.h>
#include<math.h>
#include<time.h>

const int MaxNo=50;     // Max. no. of sub-tasks
const int MaxPop=100;   // Max. limit of Population
int number;             // No. of sub-tasks to be optimized
int size;               // Population Size
int max;                // Max. no. of gen.
int xl,xr,yl,yr;        // Work-space definition
float pcross;           // Crossover probability
pinvert;               // Inversion probability
float TotalNew=0.0,TotalOld=0.0; // Total objective value of population
float dmax;
float d[MaxNo+1][MaxNo+1];
/* int stime; // For MS C++
long ltime; */
FILE *fp;
int num,kk;
int Store[MaxNo+1];
float MinValue,MaxValue,AveValue,GlobalMax=0.0;
int x[MaxNo+1];
int y[MaxNo+1];

void GetData(void);    // Gets initial parameters from user
void StartUp(void);    // Initial random population generator
int Unknown(void);     // Return a gene at random for initial population
float Fitness(int);    // Calculate objective function
void TaskCal(void);    // Sub-task calculation
void Update(void);     // Refresh the storage for initial population generation
void Generation(void); // Generate a generation
int Choose(void);      // Choose for reproduction
void Crossover(struct Gen *parent1,struct Gen *parent2,
                struct Gen *child1, struct Gen *child2); // Perform crossover
void Inversion(struct Gen *child1a, struct Gen *child2a); // Perform inversion
void Modify(struct Gen *parent1b,struct Gen *parent2b,
             struct Gen *child1b, struct Gen *child2b,int kk);   // Select best 2 as childs
void Performance(void); // Evaluate a generation
void Swap(int);        // Save the most fit chromosome
void Swap voi;
void FirstReport(void); // Report for the initial Population
void ReportS(int);     // Result of a generation
void R1(void);         // Generate h.dat
void R2(void);         // Generate v.dat
void Order(void);      // Reorder if required
struct Gen
```

215
{ int gene[MaxNo+1];
   float ObjValue;
};
struct Gen ChromeNew[MaxPop+1];
struct Gen ChromeOld[MaxPop+1];

main()
{
   int ka;
   /*ltime=time(NULL); // get current time. This is for MS C++
   stime=(unsigned) ltime/2;
   srand(stime); */ // initialize random function with sys.time
   time_t t; // For Borland C++
   srand((unsigned) time(&t)); // for Borland C++
   GetData();
   TaskCal();
   StartUp();
   Swap();
   FirstReport();
   for(ka=1;ka<=max;++ka)
   {
      Generation();
      ReportS(ka);
      Swap();
   }
   Order();
   R1();
   R2();
   return 0;
}

void GetData(void)
{
   cout<" ** *** GENETIC ALGORITHM BASED JOB OPTIMIZATION *****";
   cout<" \n\n*** WORK-SPACE ENVELOPE ***";
   cout<" \nPLEASE ENTER X-COORDINATE OF UPPER LEFT CORNER: ";
   cin>>xl;
   cout<" PLEASE ENTER Y-COORDINATE OF UPPER LEFT CORNER: ";
   cin>>yl;
   cout<" PLEASE ENTER X-COORDINATE OF LOWER RIGHT CORNER: ";
   cin>>xr;
   cout<" PLEASE ENTER Y-COORDINATE OF LOWER RIGHT CORNER: ";
   cin>>yr;
   cout<" \n*** PROGRAM PARAMETERS *** ";
   cout<" \nPLEASE ENTER NUMBER OF SUB-TASKS TO BE OPTIMIZED: ";
   cin>>number;
   while(number>MaxNo)
   {
      cout<" *** Program Limitation. Number of sub-tasks should not exceed "<MaxNo<" ***";
      cout<" \nPLEASE RE-ENTER NUMBER OF SUB-TASKS TO BE OPTIMIZED: ";
      cin>>number;
   }
}
cout<<" PLEASE ENTER POPULATION SIZE: ";
cin>>size;
while(size>MaxPop)
{
    cout<<" *** Program Limitation. Population size should not exceed "<<MaxPop<<" ***";
    cout<<"n PLEASE RE-ENTER POPULATION SIZE: ";
cin>>size;
}
cout<<" PLEASE ENTER MAXIMUM NO. OF GENERATION: ";
cin>>max;
cout<<" PLEASE ENTER CROSSOVER PROBABILITY: ";
cin>>pcross;
cout<<" PLEASE ENTER INVERSION PROBABILITY: ";
cin>>pinvert;
}

void TaskCal(void)
{
    int m,n;
    int b=1;
    float L3,L1,L2;
    dmax=(float)pow((pow((xr-xl),2)+ pow((yr-yl),2)),0.5);
    cout<<"\n";
    for(n=1; n<=number; ++n)
    {
        cout<<"enter x-comp. of location of sub-task No. "<<n<<" : ";
cin>> x[n];
        cout<<"enter y-comp. of location of sub-task No. "<<n<<" : ";
cin>> y[n];
    }
    for(m=1;m<=number;++m)
    {
        for(n=1; n<=number; ++n)
        {
            if(n!=b)
            {
                L1= (float)pow((x[n]-x[b]),2);
                L2= (float)pow((y[n]-y[b]),2);
                L3= (float) pow((L1+L2),0.5);
                d[b][n]= L3;
            }
            ++b;
        }
    }
}

void StartUp(void)
{
    int m,n;
    for(m=1;m<=size;++m)
    {
        Update();
        for(n=1; n<=number; ++n)
    }
{  // if(n==0) // These lines put 0 as first gene in every Chromosome  //  
  // ChromeNew[m].gene[0] =0;  
  // }  
  // else  
  // {  
  ChromeNew[m].gene[n]=Unknown();  
  //}  
}  
Performance();  
}  

int Unknown(void)  
{  
  int value,luck,k;  
  luck=(rand() % num)+1; // return a value bet. 1 & number  
  value=Store[luck];  
  for(k=luck+1;k<=number;++k)  
  {  
    Store[k-1]=Store[k];  
  }  
  num=num-1;  
  return value;  
}  

float Fitness(int m)  
{  
  int k;  
  float total,dt=0.0;  
  for(k=1; k<number; ++k) //  
  {  
    dt= dt+d[(ChromeNew[m].gene[k])][(ChromeNew[m].gene[k+1])];  
  }  
  total=(dmax*number)-dt;  
  return total;  
}  

int Choose(void)  
{  
  int k=1;  
  float z,z1,net=0.0;  
  z1=(float)(rand() % 99)+1;  
  z=(z1/100.0)*TotalOld;  
  while ((net-z)<0.0)  
  {  
    net=net+ChromeOld[k].ObjValue;  
    k=k+1;  
  }  
  return k-1;  
}
void Swap(void)
{
    int m,n;
    TotalOld=TotalNew;
    for(m=1;m<=size;++m)
    {
        for(n=1;n<=number;++n)
        {
            ChromeOld[m].gene[n]=ChromeNew[m].gene[n];
        }
        ChromeOld[m].Obj Value=ChromeNew[m].Obj Value;
    }
}

void Update(void)
{
    int k;
    num=number;   // num used in Unknown
    for(k=1;k<=number;++k)
    {
        Store[k]=k;
    }
}

void Generation(void)
{
    int k,Father,Mother;
    for(kk=1;kk<size;kk=kk+2)
    {
        Father=Choose();
        Mother=Choose();
        Crossover(&ChromeOld[Father],&ChromeOld[Mother],
                   &ChromeNew[kk],&ChromeNew[kk+1]);
        Inversion(&ChromeNew[kk],&ChromeNew[kk+1]);
        Modify(&ChromeOld[Father],&ChromeOld[Mother],
               &ChromeNew[kk],&ChromeNew[kk+1],kk);
    }
    Performance();
}

void Crossover(struct Gen *parent1,struct Gen *parent2,
                struct Gen *child1, struct Gen *child2)
{
    int p1,p2,p,k,count,f1,f2,f3,f4;
    float px;
    px=(float)((rand() % 100)/100.0);
    if(pcross-px>=0)
    {
        p1=(rand() % (number-4))+3;
        p2=(rand() % (number-4))+3;
        if(p1>p2)
        {

    
}
p=p2;p2=p1;p1=p;
}
int *Mating1=new int[number+1];
int *Mating2=new int[number+1];
for(k=1;k<=number;++k)
{
  Mating1[k]=0;
  Mating2[k]=0;
}
count=p1;
while(count<=p2)
{
  Mating1[(parent1->gene[count])]=parent2->gene[count];
  Mating2[(parent2->gene[count])]=parent1->gene[count];
  count=count+1;
}
for(count=1;count<=number;++count)
{
  if(count<p1 || count>p2)
  {
    k=parent2->gene[count];
    while(Mating1[k]!=0)
    {
      k=Mating1[k];
    }
    child1->gene[count]=k;
    k=parent1->gene[count];
    while(Mating2[k]!=0)
    {
      k=Mating2[k];
    }
    child2->gene[count]=k;
  }
  else
  {
    child1->gene[count]=parent1->gene[count];
    child2->gene[count]=parent2->gene[count];
  }
}
delete[]Mating1;
delete[]Mating2;
}
else
{
  for(k=1;k<=number;++k)
  {
    child1->gene[k]=parent1->gene[k];
    child2->gene[k]=parent2->gene[k];
  }
}

void Inversion(struct Gen *child1a, struct Gen *child2a)
Appendix E

```c
{  
  int p1,p2,d,k,dif;
  float px;
  px=(float) ((rand() % 1000)/1000.0);
  if((pinvert-px)>=0)
  {
    p1=(rand() % (number-1))+1;
    do{
      p2=(rand() % (number-1))+1;
    }while(p2==p1);
    if(p1>p2)
    {  
      p=p2;p2=p1;p1=p;
    }
    dif=p2-p1;
    p=p1;
    d=dif;
    int *inv=new int[dif+1];
    for(k=0;k<=dif;++k)
    {  
      inv[k]=child1a->gene[p];
      p=p+1;
    }  
    for(k=0;k<=dif;++k)
    {  
      child1a->gene[p1]=inv[d];
      p1=p1+1;
      d=d-1;
    }
    delete []inv;
  }
  px=(float) ((rand() % 1000)/1000.0);
  if((pinvert-px)>=0)
  {
    p1=(rand() % (number-1))+1;
    do{
      p2=(rand() % (number-1))+1;
    }while(p2==p1);
    if(p1>p2)
    {  
      p=p2;p2=p1;p1=p;
    }
    dif=p2-p1;
    p=p1;
    d=dif;
    int *inv=new int[dif+1];
    for(k=0;k<=dif;++k)
    {  
      inv[k]=child2a->gene[p];
      p=p+1;
    }  
    for(k=0;k<=dif;++k)
    {  
```

child2a->gene[p1]=inv[d];
p1=p1+1;
d=d-1;
}
delete []inv;
}
}

void Modify(struct Gen *parent1b, struct Gen *parent2b, 
        struct Gen *child1b, struct Gen *child2b, int kk)
{
    int k;
    float tempValue;
    child1b->ObjValue=(float)Fitness(kk);
    child2b->ObjValue=(float)Fitness(kk+1);
    tempValue=child1b->ObjValue;
    int *temp=new int[number+1];
    for(k=1;k<=number;++k)
    {
        temp[k]=child1b->gene[k];
    }
    if(child1b->ObjValue==parent1b->ObjValue || child1b->ObjValue==parent2b->ObjValue)
    {
        if(child2b->ObjValue==parent1b->ObjValue || child2b->ObjValue==parent2b->ObjValue)
        {
            goto last;
        } else
    { goto proceed; }
    }
    proceed: if(child2b->ObjValue>child1b->ObjValue)
    {
        for(k=1;k<=number;++k)
        {
            temp[k]=child2b->gene[k];
        }
        for(k=1;k<=number;++k)
        {
            child2b->gene[k]=child1b->gene[k];
        }
        for(k=1;k<=number;++k)
        {
            child1b->gene[k]=temp[k];
        }
        child1b->ObjValue=child2b->ObjValue;
        child2b->ObjValue=tempValue;
        tempValue=child1b->ObjValue;
    }
    if(child1b->ObjValue<parent1b->ObjValue || child1b->ObjValue<parent2b->ObjValue)
    {
        if(parent1b->ObjValue>parent2b->ObjValue)
        {
            for(k=1;k<=number;++k)
            {
                child1b->gene[k]=parent1b->gene[k];
            }
        }
    }
if(tempValue>parent2b->ObjValue)
{
    for(k=1;k<=number;++k)
    {
        child2b->gene[k]=temp[k];
    }
    else
    {
        for(k=1;k<=number;++k)
        {
            child2b->gene[k]=parent2b->gene[k];
        }
    }
}
else
{
    for(k=1;k<=number;++k)
    {
        child2b->gene[k]=parent2b->gene[k];
    }
    if(tempValue>parent1b->ObjValue)
    {
        for(k=1;k<=number;++k)
        {
            child2b->gene[k]=temp[k];
        }
    }
    else
    {
        for(k=1;k<=number;++k)
        {
            child2b->gene[k]=parent1b->gene[k];
        }
    }
}
if(child2b->ObjValue<parent1b->ObjValue || child2b->ObjValue<parent2b->ObjValue)
{
    if(child1b->ObjValue>parent1b->ObjValue && child1b->ObjValue>parent2b->ObjValue)
    {
        if(parent1b->ObjValue>parent2b->ObjValue)
        {
            for(k=1;k<=number;++k)
            {
                child2b->gene[k]=parent1b->gene[k];
            }
        }
        else
        {
            for(k=1;k<=number;++k)
            {
            }
child2b->gene[k]=parent2b->gene[k];
}
}
}
lstd: delete [ltemp;
}

void Performance(void)
{
int m=1,k;
MinValue=0.0,MaxValue=0.0,AveValue=0.0,TotalNew=0.0;
ChromeNew[m].ObjValue=(float)Fitness(m);
MinValue=ChromeNew[m].ObjValue;
MaxValue=ChromeNew[m].ObjValue;
TotalNew=ChromeNew[m].ObjValue;
for(m=2;m<=size;++m)
{
ChromeNew[m].ObjValue=(float)Fitness(m);
TotalNew=TotalNew+ChromeNew[m].ObjValue;
if(ChromeNew[m].ObjValue<MinValue)
{MinValue=ChromeNew[m].ObjValue;
}
if(ChromeNew[m].ObjValue>MaxValue)
{
MaxValue=ChromeNew[m].ObjValue;
if(MaxValue>GlobalMax)
{
GlobalMax=MaxValue;
Save(m);
}
}
AveValue=TotalNew/size;
}

void Save(int m)
{
int k;
for(k=1;k<=number;++k)
{
Store[k]=ChromeNew[m].gene[k];
}
}

void FirstReport(void)
{
int k;
cout<<"\nRESULT OF INITIAL POPULATION ";
cout<<"\n MinValue= "<<MinValue<< "\tMaxValue= "<<MaxValue<< "\tAveValue=
"<<AveValue<<"\n";
cout<<"MOSTFIT CHROMOSOME\n";
for(k=1;k<=number;++k)
{

void ReportS(int ka)
{
    int k;
    cout<<"n Gen= "<<ka<< " MinValue= "<<MinValue<< " MaxValue= "<<MaxValue<< " AveValue= "<<AveValue<< "n;"
    cout<<"MOSTFIT CHROMOSOMEn ";
    for(k=1;k<=number;++k)
    {
        cout<<Store[k]<<"t;"
    }
}

void Order(void)
{
    int t1, t2=number;
    if (x[Store[1]] > x[Store[number]])
    {
        int *temp1=new int[number+1];
        for(t1=1;t1<number+1;++t1)
        { temp1[t1]=Store[t1]; }
        for(t1=1;t1<number+1;++t1)
        { Store[t1]= temp1[t2];
            t2=t2-1;
        }
        delete []temp1;
    }
}

void R1(void)
{
    int k,vx=0;
    int *h=new int[number+1];
    h[1]=x[(Store[1])];
    for(k=1;k<=number;++k)
    {
        vx=x[(Store[k+1])]-x[(Store[k])];
        h[k+1]=vx;
    }
    cout<<"n *** Data in h.dat *** \n";
    for(k=1;k<=number;++k)
    {
        cout<<h[k]<<"t;"
    }
    fp= fopen("h.dat", "w");
    fprintf(fp, "%d", number); 
    for(k=1;k<=number;++k)
    {
        fprintf(fp, "%d",h[k]);
    }
}
```c
int fclose(FILE *fp);
delete []h;
}

void R2(void)
{
    int k, vy = 0;
    int *v = new int[number + 1];
    v[1] = y[(Store[1])];
    for (k = 1; k < number; ++k)
    {
        vy = y[(Store[k + 1])] - y[(Store[k])];
        v[k + 1] = vy;
    }
    cout << "\n *** Data in v.dat *** \n";
    for (k = 1; k <= number; ++k)
    {
        cout << v[k] << "t";
    }
    fp = fopen("v.dat", "w");
    fprintf(fp, "%d", number);
    for (k = 1; k <= number; ++k)
    {
        fprintf(fp, "\n%d", v[k]);
    }
    int fclose(FILE *fp);
    delete []v;
}
```
APPENDIX F

Event-Based Control Model of Drilling System

F.1 Event-Based Control Model for MM_B

Event-Based MM_B = < X_B, S_B, Y_B, δ_intB, δ_extB, λ_B, ta_B >

where

- X_B = set of external inputs
- Y_B = set of outputs
- S_B = B_B × X_B
- δ_intB((b_B1, x_B1), (b_B2, x_B2)) = (b_B2, x_B2)
- δ_extB((b_B1, x_B1), (0, x_B2)) = (b_B1, x_B2)
- λ_B((b_B1, x_B1)) = output function

- ta_B(s) = (t_2 - t_1) with t_2 > t_1 and t_2, t_1 ∈ R_0^+
- ta_B(w_n) = (w_{t_2} - w_{t_1})
  - w_{t_2} = max{ ta_A(s_n) + (ta_A(s_n) - ta_B(s_n)) } if ta_A(s_n) > ta_B(s_n) else
  - = max{ ta_CM1(s_n) } and
  - w_{t_1} = min{ ta_A(s_n) + (ta_A(s_n) - ta_B(s_n)) } if ta_A(s_n) > ta_B(s_n) else
  - = min{ ta_CM1(s_n) }

F.2 Event-Based Control Model for CM1

CM1 is a coupled model of MM_C and MM_D. The event-based based model for CM1 is, therefore, expressed in terms of parameters of MM_C and MM_D

Event-Based CM1 = < X_CM1, S_CM1, Y_CM1, δ_intCM1, δ_extCM1, λ_CM1, ta_CM1 >

where

- X_CM1 = X_C × X_D
- Y_CM1 = Y_C × Y_D
- S_CM1 = {(a, b) | a ∈ S_C, b ∈ S_D}
- δ_intCM1 = δ_intC × δ_intD
- δ_extCM1 = δ_extC × δ_extD
- λ_CM1 = λ_C × λ_D

- ta_CM1(s) = (t_2 - t_1) while t_2 > t_1 and t_2, t_1 ∈ R_0^+
- ta_CM1(w_n) = (w_{t_2} - w_{t_1})
  - w_{t_2} = max{ ta_A(s_n) } if ta_A(s_n) > ta_B(s_n) else
  - = max{ ta_B(s_n) } and
  - w_{t_1} = min{ ta_A(s_n) } if ta_A(s_n) > ta_B(s_n) else
  - = min{ ta_B(s_n) }

F.3 Event-Based Control Model for MM_C

Event-Based MM_C = < X_C, S_C, Y_C, δ_intC, δ_extC, λ_C, ta_C >

where

- X_C = set of external inputs

227
Y_C = set of outputs
S_C = B_C \times X_C where B_C = (b_{c1}, b_{c2}) as MM_C has only two state partitioning blocks
\delta_{intC}(b_{c1}, x_c) = (b_{c2}, x_c); where b_{c2} is the next boundary crossing
\delta_{extC}(b_{c1}, x_c), 0, x_c2 = (b_{c1}, x_c2)
\lambda_C(b_{c1}, x_c1) = output function

\tau_C(s) = (t_2 - t_1) with t_2 > t_1 and t_2, t_1 \in R_{0,\infty}
\tau_C(w_1) = (\min(\tau_A(s_1)), \max(\tau_A(s_1))) if \tau_A(s_1) > \tau_B(s_1) else
\quad= (\min(\tau_B(s_1)), \max(\tau_B(s_1)))
\tau_C(w_2) = (\min(\tau_D(s_2)), \max(\tau_D(s_2)))

F.4 Event-Based Control Model for MM_D
Event-Based MM_D = < X_D, Y_D, 5_D, 8_D, 5extD, 8intD, 5_D, ta_D >
where
X_D = set of external inputs; rotary speed control
Y_D = set of outputs
S_D = B_D \times X_D
\delta_{intD}(b_{d1}, x_d) = (b_{d2}, x_d); where b_{d2} is the next boundary crossing
\delta_{extD}(b_{d1}, x_d1), 0, x_d2 = (b_{d1}, x_d2)
\lambda_D(b_{d1}, x_d1) = output function
\tau_D(s) = (t_2 - t_1) with t_2 > t_1 and t_2, t_1 \in R_{0,\infty}
\tau_D(w_n) = (\min(\tau_C(s_n) + \tau_A(s_n)), \max(\tau_C(s_n) + \tau_A(s_n))
if \tau_A(s_n) > \tau_B(s_1) else
\quad= (\min(\tau_C(s_n) + \tau_B(s_n)), \max(\tau_C(s_n) + \tau_B(s_n)))
APPENDIX G

Event-Based Controller of EDS

G.1 Event-Based Control Model of CM1

CM1 is a coupled model of MM1 and MM2. During a complete work cycle, it activates twice and has two wait states (figure 6.11). The event-based control model of CM1, expressed in terms of parameters of MM1 and MM2, is as under.

\[
\text{Event-Based CM1} = \langle X_{\text{CM1}}, S_{\text{CM1}}, Y_{\text{CM1}}, \delta_{\text{IntCM1}}, \delta_{\text{ExtCM1}}, \lambda_{\text{CM1}}, ta_{\text{CM1}} \rangle
\]

where

- \( X_{\text{CM1}} = X_1 \times X_2 \)
- \( Y_{\text{CM1}} = Y_1 \times Y_2 \)
- \( S_{\text{CM1}} = \{ (a, b) \mid a \in S_1, b \in S_2 \} \)
- \( \delta_{\text{IntCM1}} = \delta_{\text{Int1}} \times \delta_{\text{Int2}} \)
- \( \delta_{\text{ExtCM1}} = \delta_{\text{Ext1}} \times \delta_{\text{Ext2}} \)
- \( \lambda_{\text{CM1}} = \lambda_1 \times \lambda_2 \)

\[
ta_{\text{CM1}(s)} = (t_2 - t_1) \text{ while } t_2 > t_1 \text{ and } t_2, t_1 \in R^+; \text{ time window}
\]

\[
ta_{\text{CM1}(w_1)} = [\min\{ta_A(s)\}, \max\{ta_A(s)\}]; \text{ first wait state (figure 6.9)}
\]

\[
ta_{\text{CM1}(w_2)} = [\min\{ta_{CM2}(s_2) + ta_{CM4}(s_1 + s_2) + ta_{CM2}(s_1) - ta_{CM1}(s_2)\},
\max\{ta_{CM2}(s_2) + ta_{CM4}(s_1 + s_2) + ta_{CM2}(s_1) - ta_{CM1}(s_2)\}]; \text{ second wait state}
\]

CM1 does not need to be activated during the last work cycle in which the last drilled plate is brought from the work area to the storehouse. \( ta_{\text{CM1}(w_2)} \) thus becomes \( \infty \) at the end of the second last cycle of system operation making CM1 inactive during the last cycle of operation.

G.1.1 Event-Based Control Model of MM1

Event-Based MM1 = \( \langle X_1, S_1, Y_1, \delta_{\text{Int1}}, \delta_{\text{Ext1}}, \lambda_1, ta_1 \rangle \)

where

- \( X_1 \) = set of external inputs; speed control of MM1
- \( Y_1 \) = set of outputs
- \( S_1 \) = \( B_1 \times X_1 \) where \( B_1 = (b_{11}, b_{12}) \); MM1 has two state partitioning boundaries
- \( \delta_{\text{Int1}}(b_{11}, x_1) = (b_{12}, x_1) \) where \( b_{12} \) is the next boundary crossing
- \( \delta_{\text{Ext1}}((0, 0), 0, x_1) = (b_{11}, x_1) \)
- \( \lambda_1(b_{11}, x_1) = \text{ output function; generates appropriate signals} \)

\[
ta_1(s) = (t_2 - t_1) \text{ with } t_2 > t_1 \text{ and } t_2, t_1 \in R^+; \text{ time window}
\]

The three wait states of MM1, as obvious from figure 6.9 and 6.11, are:

- \( ta_1(w_1) = [\min\{ta_A(s)\}, \max\{ta_A(s)\}] \)
- \( ta_1(w_2) = [\min\{ta_A(s)\}, \max\{ta_A(s)\}] \)
- \( ta_1(w_3) = [\min\{ta_{CM1}(w_2) + ta_A(s_2) - ta_1(s_2)\}, \max\{ta_{CM1}(w_2) + ta_A(s_2) - ta_1(s_2)\}] \)

During the second last cycle of operation, \( ta_1(w_3) \) becomes \( \infty \), i.e., MM1 does not activate any further.

G.1.2 Event-Based Control Model of MM2

Event-Based MM2 = \( \langle X_2, S_2, Y_2, \delta_{\text{Int2}}, \delta_{\text{Ext2}}, \lambda_2, ta_2 \rangle \)

where
Appendix G

230

• \( X_2 \) = set of external inputs; speed control of MM2
• \( Y_2 \) = set of outputs
• \( S_2 = B_2 \times X_2 \) where \( B_2 = (b_{21}, b_{22}) \); MM2 has two state partitioning boundaries
• \( \delta_{\text{int}2}(b_{21}, x_2) = (b_{22}, x_2) \) where \( b_{22} \) is the next boundary crossing
• \( \delta_{\text{ext}2}(b_{21}, x_2) = (b_{21}, x_2) \)
• \( \lambda_2(b_{21}, x_2) = \) output function; generates appropriate signals

\[ \begin{align*}
\text{ta}_2(s) &= (t_2 - t_1) \quad \text{with } t_2 > t_1 \text{ and } t_2, t_1 \in R_0^+; \text{ time window} \\
\text{ta}_2(w_1) &= [\min\{\text{ta}_1(s_1 + s_2)\}, \max\{\text{ta}_1(s_1 + s_2)\}] \; ; \text{first wait state} \quad \text{figure 6.9 and 6.11} \\
\text{ta}_2(w_2) &= [\min\{\text{ta}_4(s) + \text{ta}_A(s)\}, \max\{\text{ta}_4(s) + \text{ta}_A(s)\}] \; ; \text{second wait state} \\
\text{ta}_2(w_3) &= \text{ta}_{\text{CM1}}(w_2) \; ; \text{third wait state} \\
\text{ta}_2(w_3) &= \infty \; \text{in the second last cycle of operation, i.e., MM2 does not activate any further.}
\end{align*} \]

G.2 Event-Based Control Model of CM2

CM2 is a coupled model of MM3, MM4 and CM21. During a work cycle, it activates three times and has four wait states (figure 6.12). The event-based control model for CM2 is expressed in terms of parameters of MM3, MM4 and CM21.

Event-Based CM2 = \(< X_{\text{CM2}}, S_{\text{CM2}}, Y_{\text{CM2}}, S_{\text{ntCM2}}, S_{\text{extCM2}}, A_{\text{CM2}}, \text{ta}_{\text{CM2}} > \)

where

• \( X_{\text{CM2}} = X_3 \times X_4 \times X_{\text{CM21}} \)
• \( Y_{\text{CM2}} = Y_3 \times Y_4 \times Y_{\text{CM21}} \)
• \( S_{\text{CM2}} = \{ (a, b, c) \mid a \in S_3, b \in S_4, c \in S_{\text{CM21}} \} \)
• \( \delta_{\text{intCM2}} = \delta_{\text{int}3} \times \delta_{\text{int}4} \times \delta_{\text{CM21}} \)
• \( \delta_{\text{extCM2}} = \delta_{\text{ext}3} \times \delta_{\text{ext}4} \times \delta_{\text{CM21}} \)
• \( \lambda_{\text{CM2}} = \lambda_3 \times \lambda_4 \times \lambda_{\text{CM21}} \)

\[ \begin{align*}
\text{ta}_{\text{CM2}}(s) &= (t_2 - t_1) \quad \text{while } t_2 > t_1 \text{ and } t_2, t_1 \in R_0^+; \text{ time window} \\
\text{ta}_{\text{CM2}}(w_1) &= [\min\{\text{ta}_{\text{CM4}}(s_1)\}, \max\{\text{ta}_{\text{CM4}}(s_1)\}] \; ; \text{fourth wait state} \\
\text{ta}_{\text{CM2}}(w_2) &= [\min\{\text{ta}_{\text{CM4}}(s_2)\}, \max\{\text{ta}_{\text{CM4}}(s_2)\}] \; ; \text{fourth wait state} \\
\text{ta}_{\text{CM2}}(w_2) &= \infty \; \text{in the last cycle of operation, i.e., CM2 stops further functioning.}
\end{align*} \]

G.2.1 Event-Based Control Model of MM3

Event-Based MM3 = \(< X_3, S_3, Y_3, \delta_{\text{int}3}, \delta_{\text{ext}3}, \lambda_3, \text{ta}_3 > \)

where

• \( X_3 \) = set of external inputs; speed control of MM3
• \( Y_3 \) = set of outputs
• \( S_3 = B_3 \times X_3 \) where \( B_3 \) depends on the number and location of holes to be drilled
• \( \delta_{\text{int}3}(b_{31}, x_3) = (b_{32}, x_3) \) where \( b_{32} \) is the next boundary crossing
• \( \delta_{\text{ext}3}(b_{31}, x_3) = (b_{31}, x_3) \)
• \( \lambda_3(b_{31}, x_3) = \) output function; generates appropriate signals

\[ \begin{align*}
\text{ta}_3(s) &= (t_2 - t_1) \quad \text{with } t_2 > t_1 \text{ and } t_2, t_1 \in R_0^+; \text{ time window}
\end{align*} \]

MM3 has 4+N wait states where N is the total number of holes to be drilled (figure 6.12). Four of these wait states are the same as the wait states of CM2 while the other N wait states have the general expression of
• \( t_a(s_N) = [\min\{t_a(s_N)+t_a(s_N-1)+t_a(s_N-2)\}, \max\{t_a(s_N)+t_a(s_N-1)+t_a(s_N-2)\}] \)

### G.2.2 Event-Based Control Model of MM4

**Event-Based MM4** = \(<X_4, S_4, Y_4, \delta_{\text{int}4}, \delta_{\text{ext}4}, \lambda_4, t_a_4>>

where
- \(X_4\) = set of external inputs; speed control of MM4
- \(Y_4\) = set of outputs
- \(S_4\) = \(B_4 \times X_4\) where \(B_4\) depends on the number and location of holes to be drilled
- \(\delta_{\text{int}4}(b_{41}, x_4) = (b_{42}, x_4)\) where \(b_{42}\) is the next boundary crossing
- \(\delta_{\text{ext}4}(b_{41}, x_{41}), 0, x_{42} = (b_{41}, x_{42})\)
- \(\lambda_4(b_{41}, x_4) = \text{output function; generates appropriate signals}\)

- \(t_a_4(s) = (t_2 - t_1)\) with \(t_2 > t_1\) and \(t_2, t_1 \in \mathbb{R}^+\); time window

MM4 has \(3+N\) wait states where \(N\), as before, is the number of holes to be drilled. These wait states can be expressed as (figure 6.12)

- \(t_a_4(w_l) = [\min\{t_a_4(w_l)+t_a_3(s_{N-1})\}, \max\{t_a_4(w_l)+t_a_3(s_{N-1})\}]\) if \(t_a_3(s_{N-1}) > t_a_4(s_{N-1})\) else
- \(t_a_4(w_l) = [\min\{t_a_4(w_l)+t_a_3(s_{N-1})\}, \max\{t_a_4(w_l)+t_a_3(s_{N-1})\}]\)

The other \(N\) wait states, like the wait states of MM3, can be expressed as
- \(t_a_4(w_N) = [\min\{t_a_4(s_N)+t_a_3(s_{N-1})\}, \max\{t_a_4(s_N)+t_a_3(s_{N-1})\}]\)

### G.2.3 Event-Based Control Model for CM21

CM21 is a sub-coupled model of MM5 and MM6 and can be expressed as

**Event-Based CM21** = \(<X_{CM21}, S_{CM21}, Y_{CM21}, \delta_{\text{int}CM21}, \delta_{\text{ext}CM21}, \lambda_{CM21}, t_a_{CM21}>>

- \(X_{CM21} = X_5 \times X_6\)
- \(Y_{CM21} = Y_5 \times Y_6\)
- \(S_{CM21} = \{(a, b) | a \in S_5, b \in S_6\}\)
- \(\delta_{\text{int}CM21} = \delta_{\text{int}5} \times \delta_{\text{int}6}\)
- \(\delta_{\text{ext}CM21} = \delta_{\text{ext}5} \times \delta_{\text{ext}6}\)
- \(\lambda_{CM21} = \lambda_5 \times \lambda_6\)

- \(t_a_{CM21}(s) = (t_2 - t_1)\) while \(t_2 > t_1\) and \(t_2, t_1 \in \mathbb{R}^+\)

CM21, like MM4, has 3+\(N\) wait states. These wait state are given as (figure 6.13)

- \(t_a_{CM21}(w_l) = [\min\{t_a_4(w_l)+t_a_3(s_{N-1})\}, \max\{t_a_4(w_l)+t_a_3(s_{N-1})\}]\) if \(t_a_3(s_{N-1}) > t_a_4(s_{N-1})\) else \(t_a_{CM21}(w_l) = [\min\{t_a_4(w_l)+t_a_3(s_{N-1})\}, \max\{t_a_4(w_l)+t_a_3(s_{N-1})\}]\)

- \(t_a_{CM21}(w_N) = [\min\{t_a_4(s_N)\}, \max\{t_a_4(s_N)\}]\)

### G.2.3.1 Event-Based Control Model of MM5

**Event-Based Control Model of MM5** = \(<X_5, S_5, Y_5, \delta_{\text{int}5}, \delta_{\text{ext}5}, \lambda_5, t_a_5>>

where
- \(X_5\) = set of external inputs; speed control of MM5
- \(Y_5\) = set of outputs
- \(S_5 = B_5 \times X_5\) where \(B_5 = (b_{51}, b_{52})\); MM5 has two partitioning boundaries
• \( \delta_{\text{int}5}(b_{51}, x_{5}) = (b_{52}, x_{5}) \) where \( b_{52} \) is the next boundary crossing

• \( \delta_{\text{ext}5}((b_{51}, x_{51}), 0, x_{52}) = (b_{51}, x_{52}) \)

• \( \lambda_{5}(b_{51}, x_{51}) = \) output function; generates appropriate signals

• \( t_{a5}(s) = (t_{2} - t_{1}) \) with \( t_{2} > t_{1} \) and \( t_{2}, t_{1} \in \mathbb{R}_{0,\infty}^{+} \); time window

MM5 has also 3+N wait states defined as (figure 6.13)

• \( t_{a5}(w_{j}) = t_{aCM21}(w_{j}) \)
• \( t_{a5}(w_{2}) = t_{aCM4}(s_{1}) \)
• \( t_{a5}(w_{3}) = t_{aCM4}(s_{2}) \)
• \( t_{a5}(w_{N}) = \left[ \min\{t_{a5}(s_{N})\}, \max\{t_{a5}(s_{N})\} \right] \)

G.2.3.2 Event-Based Control Model of MM6

Event-Based MM6 = \( < X_{6}, S_{6}, Y_{6}, \delta_{\text{int}6}, \delta_{\text{ext}6}, \lambda_{6}, t_{a6} > \)

where

• \( X_{6} \) is set of external inputs; speed control of MM6
• \( Y_{6} \) is set of outputs
• \( S_{6} = B_{6} \times X_{6} \) where \( B_{6} = (b_{61}, b_{62}) \); MM6 has two partitioning boundaries
• \( \delta_{\text{int}6}(b_{61}, x_{6}) = (b_{62}, x_{6}) \); where \( b_{62} \) is the next boundary crossing
• \( \delta_{\text{ext}6}(b_{61}, x_{61}), 0, x_{62} = (b_{61}, x_{62}) \)
• \( \lambda_{6}(b_{61}, x_{6}) = \) output function; generates appropriate signals

• \( t_{a6}(s) = (t_{2} - t_{1}) \) with \( t_{2} > t_{1} \) and \( t_{2}, t_{1} \in \mathbb{R}_{0,\infty}^{+} \); time window

The 3+N wait states of MM6 are given as (figure 6.13)

• \( t_{a6}(w_{j}) = t_{aCM21}(w_{j}) \)
• \( t_{a6}(w_{2}) = t_{aCM4}(s_{1}) \)
• \( t_{a6}(w_{3}) = t_{aCM4}(s_{2}) \)
• \( t_{a6}(w_{N}) = \left[ \min\{t_{a6}(s_{N})\}, \max\{t_{a6}(s_{N})\} \right] \)

G.3 Event-Based Control Model of CM3

CM3 is a coupled model of MM7 and MM8. It activates twice during a complete work cycle and has three wait states (figure 6.14). The event-based control model of CM3 is given as under.

Event-Based CM3 = \( < X_{CM3}, S_{CM3}, Y_{CM3}, \delta_{\text{int}CM3}, \delta_{\text{ext}CM3}, \lambda_{CM3}, t_{aCM3} > \)

where

• \( X_{CM3} = X_{7} \times X_{8} \)
• \( Y_{CM3} = Y_{7} \times Y_{8} \)
• \( S_{CM3} = \{(a, b) \mid a \in S_{7}, b \in S_{8}\} \)
• \( \delta_{\text{int}CM3} = \delta_{\text{int}7} \times \delta_{\text{int}8} \)
• \( \delta_{\text{ext}CM3} = \delta_{\text{ext}7} \times \delta_{\text{ext}8} \)
• \( \lambda_{CM3} = \lambda_{7} \times \lambda_{8} \)

• \( t_{aCM3}(s) = (t_{2} - t_{1}) \) while \( t_{2} > t_{1} \) and \( t_{2}, t_{1} \in \mathbb{R}_{0,\infty}^{+} \)

• \( t_{aCM3}(w_{j}) = t_{aCM21}(w_{j}) \); first wait state
• \( t_{aCM3}(w_{2}) = t_{aCM22}(w_{2}) \); second wait state
• \( t_{aCM3}(w_{3}) = \left[ \min\{t_{aCM4}(s_{1}+s_{2})+t_{aCM2}(s_{2}+s_{3})-t_{aCM3}(s_{2})\}, \right. \)
\[ \left. \max\{t_{aCM4}(s_{1}+s_{2})+t_{aCM2}(s_{2}+s_{3})-t_{aCM3}(s_{2})\} \right] \]; third wait state

During the first cycle of operation, CM3 is not required to be activated as there is no drilled plate in the work area to be fetched into the storehouse.
Appendix G

G.3.1 Event-Based Control Model of MM7

Event-Based MM7 = < X7, S7, Y7, δint7, δext7, λ7, ta7 >

where

- X7 = set of external inputs; speed control of MM7
- Y7 = set of outputs
- S7 = B7 x X7 where B7 = (b71, b72); MM7 has two state partitioning boundaries
- δint7(b71, x7) = (b72, x7) where b72 is the next boundary crossing
- δext7((b71, x7), 0, x72) = (b71, x72)
- λ7(b71, x7) = output function; generates appropriate signals
- ta7(s) = (t2 - t1) with t2 > t1 and t2, t1 ∈ R+; time window

The three wait states of MM7 are given as

- ta7(w1) = max{ta7(s) + ta8(s1) + ta7(s1)}
- ta7(w2) = [ min{ta7(s2) + taCM4(w3)}, max{ta7(s2)} ]
- ta7(w3) = [ min{ta7(s3) + taCM4(w3)}, max{ta7(s3)} ]

G.3.2 Event-Based Control Model of MM8

Event-Based MM8 = < X8, S8, Y8, δint8, δext8, λ8, ta8 >

where

- X8 = set of external inputs; speed control of MM8
- Y8 = set of outputs
- S8 = B8 x X8 where B8 = (b81, b82); MM8 has two state partitioning boundaries
- δint8(b81, x8) = (b82, x8) where b82 is the next boundary crossing
- δext8((b81, x8), 0, x82) = (b81, x82)
- λ8(b81, x8) = output function; generates appropriate signals
- ta8(s) = (t2 - t1) with t2 > t1 and t2, t1 ∈ R+; time window

MM8 also has following three wait states.

- ta8(w1) = max{ta7(s) + ta8(s1) + ta7(s1)}
- ta8(w2) = [ min{ta7(s2)}, max{ta7(s2)} ]
- ta8(w3) = taCM4(w3)

G.4 Event-Based Control Model of CM4

CM4 is a coupled model of MM9 and MM10. It activates twice during a complete work cycle and has two wait states (figure 6.15). The event-based control model of CM4 is expressed as under.

Event-Based CM4 = < XCM4, SCM4, YCM4, δintCM4, δextCM4, λCM4, taCM4 >

where

- XCM4 = X9 x X10
- YCM4 = Y9 x Y10
- SCM4 = {[(a, b) | a ∈ S9, b ∈ S10} 
- δintCM4 = δint9 x δint10
- δextCM4 = δext9 x δext10
- λCM4 = λ9 x λ10
- taCM4(s) = (t2 - t1) while t2 > t1 and t2, t1 ∈ R+; time window
Appendix G

The wait states of CM4 are given as (figure 6.9 and 6.15):
- \[ \text{ta}_{CM4}(w_1) = \left[ \min \{ \text{ta}_{CM2}(w_1+s_1+w_2+s_2) \}, \max \{ \text{ta}_{CM2}(w_1+s_1+w_2+s_2) \} \right] \]
- \[ \text{ta}_{CM4}(w_2) = \left[ \min \{ \text{ta}_{CM2}(s_3) \}, \max \{ \text{ta}_{CM2}(s_3) \} \right] \]

G.4.1 Event-Based Control Model of MM9

Event-Based MM9 = \(< X_9, S_9, Y_9, \delta_{\text{int}_9}, \delta_{\text{ext}_9}, \lambda_9, t_a_9 >\>

where
- \( X_9 \) = set of external inputs; speed control of MM9
- \( Y_9 \) = set of outputs
- \( S_9 = B_9 \times X_9 \) where \( B_9 = (b_{91}, b_{92}) \); MM9 has two state partitioning boundaries
- \( \delta_{\text{int}_9}(b_{91}, x_9) = (b_{92}, x_9) \) where \( b_{92} \) is the next boundary crossing
- \( \delta_{\text{ext}_9}(b_{91}, x_9) = (b_{91}, x_9) \)
- \( \lambda_9(b_{91}, x_9) = \) output function; generates appropriate signals
- \( t_a_9(s_9) = (t_2 - t_1) \) with \( t_2 > t_1 \) and \( t_2, t_1 \in R^+ \); time window

The two wait states of MM9 are given as (figure 6.15):
- \[ \text{ta}_9(w_1) = \text{ta}_{CM4}(w_1) \]
- \[ \text{ta}_9(w_2) = \left[ \min \{ \text{ta}_{CM2}(s_3) \}, \max \{ \text{ta}_{CM2}(s_3) \} \right] \]

G.4.2 Event-Based Control Model of MM10

Event-Based MM10 = \(< X_{10}, S_{10}, Y_{10}, \delta_{\text{int}_{10}}, \delta_{\text{ext}_{10}}, \lambda_{10}, t_a_{10} >\>

where
- \( X_{10} \) = set of external inputs; speed control of MM10
- \( Y_{10} \) = set of outputs
- \( S_{10} = B_{10} \times X_{10} \) where \( B_{10} = (b_{101}, b_{102}) \); MM10 has two state partitioning boundaries
- \( \delta_{\text{int}_{10}}(b_{101}, x_{10}) = (b_{102}, x_{10}) \) where \( b_{102} \) is the next boundary crossing
- \( \delta_{\text{ext}_{10}}(b_{101}, x_{10}) = (b_{101}, x_{10}) \)
- \( \lambda_{10}(b_{101}, x_{10}) = \) output function; generates appropriate signals
- \( t_a_{10}(s_{10}) = (t_2 - t_1) \) with \( t_2 > t_1 \) and \( t_2, t_1 \in R^+ \); time window

The three wait states of MM10 (figure 6.15) are given as:
- \[ \text{ta}_{10}(w_1) = \left[ \min \{ \text{ta}_9(w_1+s_1) \}, \max \{ \text{ta}_9(w_1+s_1) \} \right] \]
- \[ \text{ta}_{10}(w_2) = \left[ \min \{ \text{ta}_{CM2}(s_3) \}, \max \{ \text{ta}_{CM2}(s_3) \} \right] \]
- \[ \text{ta}_{10}(w_3) = \left[ \min \{ \text{ta}_9(s_2) \}, \max \{ \text{ta}_9(s_2) \} \right] \]

G.5 Event-Based Control Model of MMA

Event-Based MMA = \(< X_A, S_A, Y_A, \delta_{\text{int}_A}, \delta_{\text{ext}_A}, \lambda_A, t_a_A >\>

where
- \( X_A \) = set of external inputs; speed control of MMA
- \( Y_A \) = set of outputs
- \( S_A = B_A \times X_A \) where \( B_A = (b_{A1}, b_{A2}) \); MMA has two state partitioning boundaries
- \( \delta_{\text{int}_A}(b_{A1}, x_A) = (b_{A2}, x_A) \)
- \( \delta_{\text{ext}_A}(b_{A1}, x_A) = (b_{A1}, x_A) \)
- \( \lambda_A(b_{A1}, x_A) = \) output function; generates appropriate signals
- \( t_a_A(s_A) = (t_2 - t_1) \) with \( t_2 > t_1 \) and \( t_2, t_1 \in R^+ \); time window

MMA has two wait states given by (figure 6.9)
\[ \left[ \left( (\zeta_{s+1})^{1} \right) \left( \omega_{1} \right) \left( \omega_{2} \right) \left( \omega_{3} \right) \right] \max \left( \left( \zeta_{s+1} \right) \left( \omega_{1} \right) \left( \omega_{2} \right) \left( \omega_{3} \right) \right) = (\zeta_{s+1})^{1} \omega_{1} \omega_{2} \omega_{3} \]