ELSEVIER

Computers in Industry 35 (1998) 121–147

COMPUTERS IN INDUSTRY

Reliable automated manufacturing system design based on SMT framework

W.-H. Jeng [a,b,*], G.R. Liang [b]

[a] Department of Industrial Engineering and Management, Ming Hsin Institute of Technology 1, Hsin-Hsing Road, Hsin-Fong, Hsin-Chu, Taiwan

[b] Institute of Industrial Engineering National Chiao Tung University 1001, Ta-Hsueh Road, Hsin-Chu, Taiwan

Accepted 18 July 1997

Abstract

In this paper, we propose a design methodology based on SMT (Supervisor–Monitor–Troubleshooter) framework to build a reliable AMS. In SMT framework, supervisors dictate the control logic of AMS; monitors detect faults if they occur, and troubleshooters conduct fault diagnosis and recovery. The methodology includes four stages for building an AMS: supervisor design, monitor design, troubleshooter design, and implementation. Moreover, an AMS called *Mold Filling System* is employed to demonstrate the effectiveness of the methodology's design and implementation. The methodology proposed here allows a computer system to supervise, monitor, and troubleshoot a remote physical system with significantly enhanced reliability. © 1998 Elsevier Science B.V.

Keywords: Automated manufacturing system; Supervisor; Monitor; Troubleshooter

1. Introduction

An automated manufacturing system (AMS) usually consists of a number of devices controlled by computers to manufacture products. AMS plays a critical role in chemical processing, parts manufacturing, and product assembly. As the global competitiveness increases, a more reliable AMS becomes desirable. To build such a system, designers typically perform the following: functional analysis, process control design, system verification, fault-proof planning, error recovery, and final implementation. AMS specifications are obtained though functional analysis. Process control design gives system control logic according to the specifications. In addition, system verification justifies the control logic. Moreover, fault-proof planning maintains the system availability, and error recovery solves faults.

Previous literature distinguishes four relevant AMS design issues, i.e., supervisory control [1,2], monitoring [3,4], fault diagnosis [5–7], and error recovery [8–11]. To our best knowledge, no work has been presented with an integrated design method for supervisory control, monitoring, and troubleshooting.

[*] Corresponding author. Fax: +886-3-5595142; e-mail: whjeng@mhit.edu.tw

In this study, we propose a design methodology based on SMT (Supervisor–Monitor–Troubleshooter) framework [12] to build a reliable AMS. The methodology integrates the design work in functional analysis, process control, system verification, monitoring, fault diagnosis, and error recovery. Through the methodology, an AMS can be systematically implemented with real time capability in control, monitoring, and troubleshooting via computer network.

The rest of this paper is organized as follows. Section 2 describes SMT framework and introduces the proposed methodology. Sections 3–5 present the design of supervising, monitoring, and troubleshooting, respectively. Section 6 describes how to implement an AMS according to the design methodology. Concluding remarks are finally made in Section 7.

2. An integrated design methodology

2.1. SMT framework

In SMT framework (Fig. 1), an AMS consists of a physical manufacturing system (PMS) and a computer system. The PMS contains controllers and physical devices (PDs) to perform physical production. The computer system includes a decision system and a virtual manufacturing system (VMS). Supervisors, monitors, and troubleshooters constitute the decision system. The VMS is an image of the PMS and contains virtual devices (VDs). Each VD may have its own output, input, and state. The output maps to a port of the controller which drives actuators; the input receives the PMS responses from sensors; and the state stores inferred information regarding the PMS.

An operator or high-level management system gives instructions through system interface to the supervisors. The supervisors then execute the instructions and govern the PMS by updating the outputs of VDs. The situation of PMS is fed back to the inputs and further shown by performance indicators.

Monitors observe the behavior of VMS by referring to the inputs, states, and outputs of VDs. When the behavior is normal, monitors pass the control to supervisors. Any monitor detecting abnormal behavior switches the control immediately to a troubleshooter which conducts fault diagnosis and recovery right away. Thus, the fault does not propagate and cause further damage to the system. Diagnosis locates the probable fault sources and then informs the operators with an alarm message. Without the assists from the operator, the troubleshooter can automatically recover some faults which are called recoverable faults. The other faults are the unrecoverable

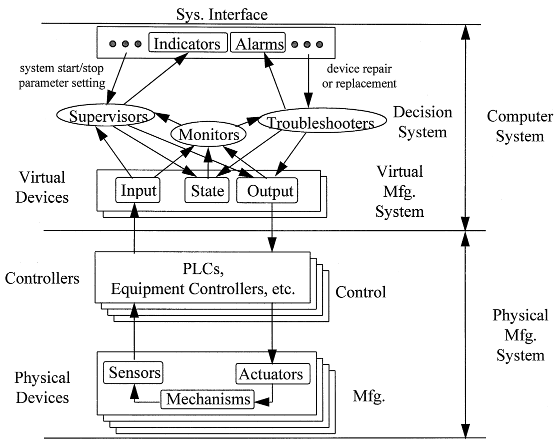


Fig. 1. SMT framework.

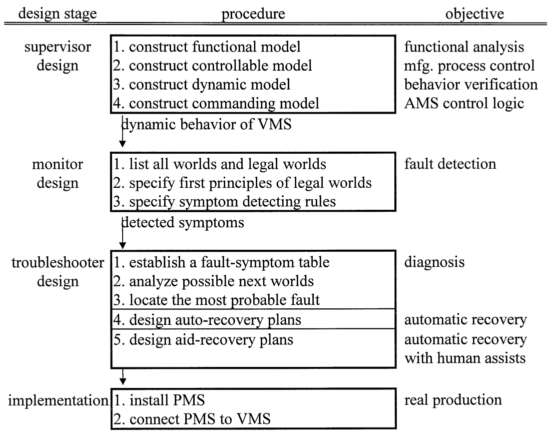

design stage
procedure
objective
supervisor design
1. construct functional model
2. construct controllable model
3. construct dynamic model
4. construct commanding model
functional analysis
mfg. process control
behavior verification
AMS control logic
dynamic behavior of VMS
monitor design
1. list all worlds and legal worlds
2. specify first principles of legal worlds
3. specify symptom detecting rules
fault detection
detected symptoms
troubleshooter design
1. establish a fault-symptom table
2. analyze possible next worlds
3. locate the most probable fault
4. design auto-recovery plans
5. design aid-recovery plans
diagnosis
automatic recovery
automatic recovery with human assists
implementation
1. install PMS
2. connect PMS to VMS
real production

Fig. 2. A design methodology for building a reliable AMS.

ones. To recover an unrecoverable fault, the operator is informed to eliminate the fault source, such as to repair or replace a device. Once the fault source is eliminated, he may instruct the troubleshooter to continue the recovery. Such a continuation can save human recovery efforts and avert manual recovery errors.

2.2. Design methodology

In the methodology, an AMS is built in four stages: supervisor design, monitor design, troubleshooter design, and implementation stage, as shown in Fig. 2. Currently, the methodology concentrates on batch process and discrete manufacturing; details regarding continuous control such as PID dynamics are not involved. The supervisor design stage attempts to obtain the control logic of an AMS. Four models are constructed: a functional model to represent the material flows and specify a manufacturing process, a controllable model to control the manufacturing process, a dynamic model to verify VMS's dynamic behavior, and a commanding model to execute the control logic. Each model has a hierarchical structure. The model construction process utilizes Hierarchy Transformation Method (HTM) [13] which has been presented for AMS specification and verification.

The monitor design stage aims to detect abnormal behavior or symptoms in an AMS. VMS's verified dynamic behavior is used as normal system dynamics. All possible states of the VMS are regarded as worlds. According to the system dynamics, only some worlds are legal. Symptom detecting rules are designed to detect wrong world transitions and abnormal behavior of PMS.

The troubleshooter design stage considers diagnosis and recovery. Faults are located through diagnosis. Each fault is then solved by a recovery plan. Auto-recovery plans are designed to automatically solve recoverable faults without human intervention. Aid-recovery plans are designed to continue the recovery, after unrecoverable fault sources have been eliminated.

The implementation stage aims to realize the AMS. A PMS is installed as specified by the VMS and is connected to the VMS via a handshaking protocol. Thus, the PMS can be remotely controlled and troubleshot by a computer system.

3. Supervisor design

Supervisors are constructed by a four-step procedure from functional model to commanding model as shown in Fig. 2. To illustrate supervisor construction, an AMS called *Mold Filling System* [13,14] is employed as

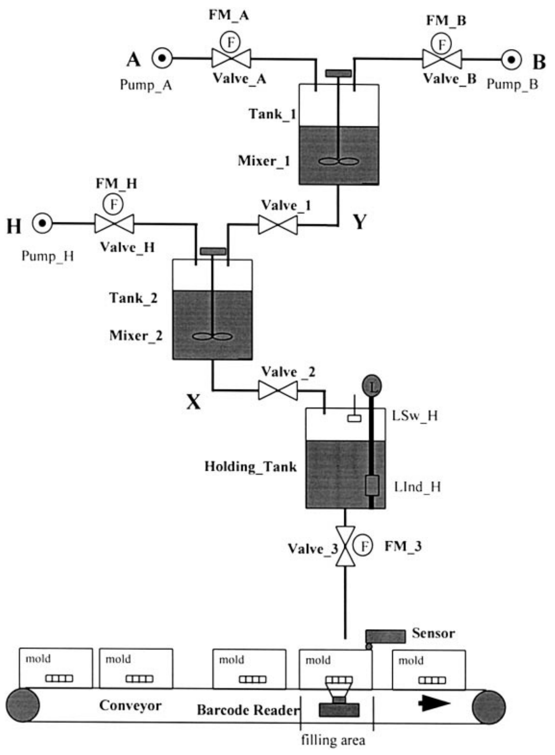


Fig. 3. Mold Filling System.

shown in Fig. 3. The system produces batches of material X, fills X into molds according to the volume message from a barcode reader, and ships out the filled molds. Material X is prepared by mixing Y and H. Material Y is prepared by mixing A and B. This AMS contains typical batch process and discrete manufacturing. Joannis and Krieger [14] first modeled this AMS via an object-oriented approach. Also, Liang and Hong [13] specified and verified the same AMS through HTM. Herein, we illustrate how to utilize HTM for supervisor construction.

3.1. Functional model construction

A functional model gives the specifications of an AMS including material flows and manufacturing functions. The model is represented by a functional hierarchy [13] as

$$\Sigma\text{FM} = \{\text{FM}_i(\text{F,M,In,Out}) | i \text{ is an index}\}$$

where F is a set of manufacturing functions or functional blocks; M is a set of material flows; both input function In:F → M and output function Out:F → M specify the graphic relations between F and M.

For instance, two main functions are included in *Mold Filling System*. FM_1 is to prepare X, and FM_2 is to fill mold and ship out the mold. Figs. 4 and 5 show these two functions, respectively. For FM_1, the representation is as follows:

Model:	$FM_1(F,M,In,Out)$
F:	$\{F11,F12,F13\}$
M:	$\{A, B, H, Y, X_{(F12,F13)}, X_{(F13,F2)}\}$
In:	$In(F11) = \{A, B\}$
	$In(F12) = \{Y,H\}$
	$In(F13) = \{X_{(F12,F13)}\}$
Out:	$Out(F11) = \{Y\}$
	$Out(F12) = \{X_{(F12,F13)}\}$
	$Out(F13) = \{X_{(F13,F2)}\}$

In a functional model, functions and their relations are specified by manufacturing activities and material flows. Each function is drawn as a box and the material flows are drawn as thick arrows. The details of each

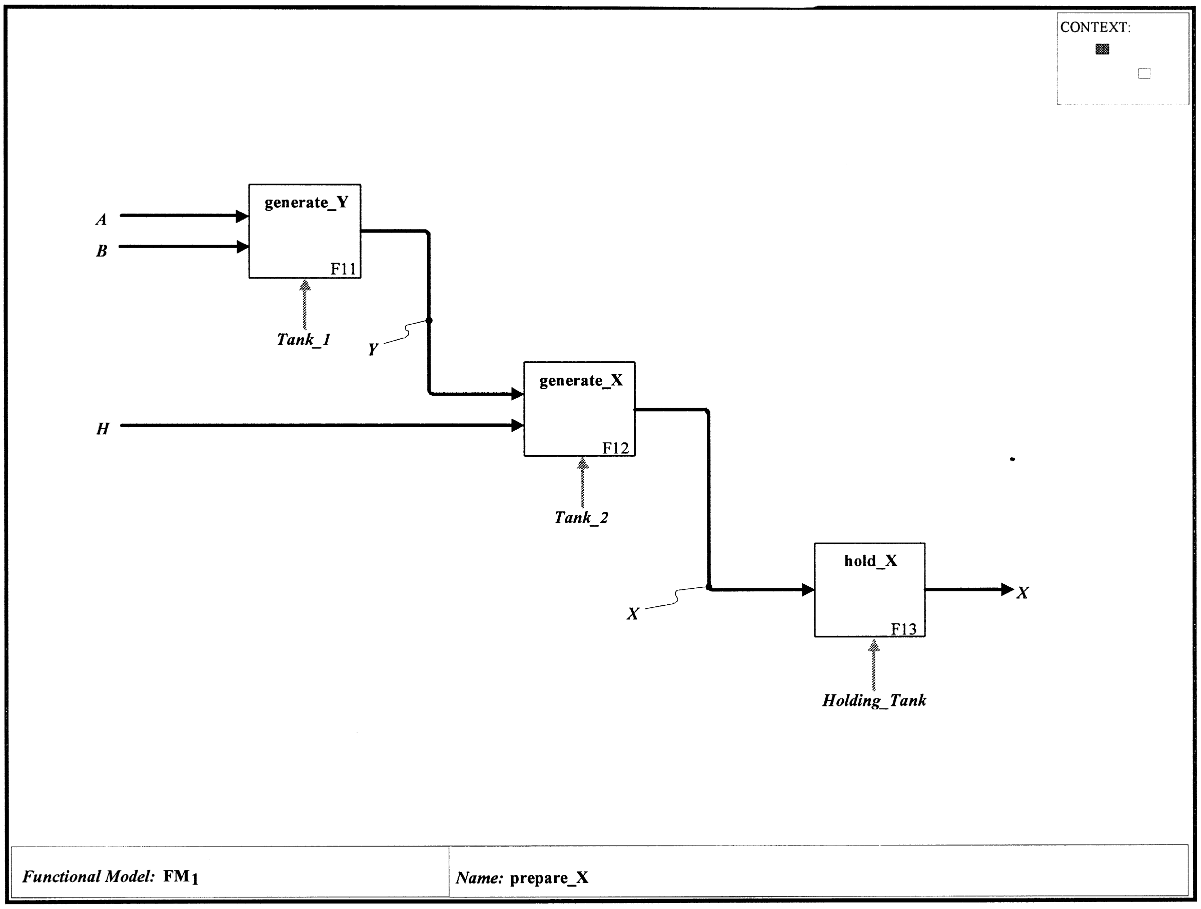


Fig. 4. Manufacturing function prepare _X (FM_1) in functional hierarchy.

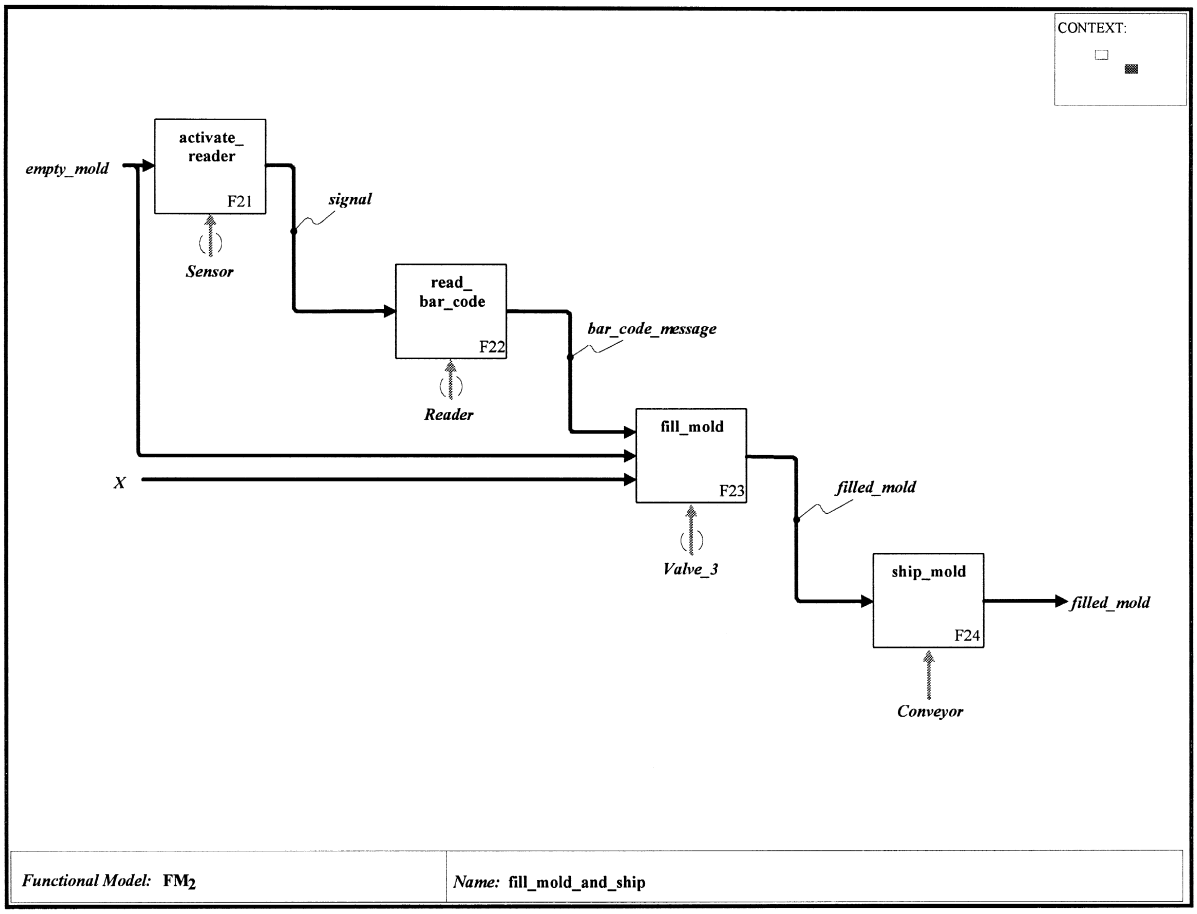


Fig. 5. Manufacturing function fill_mold_and_ship (FM_2) in functional hierarchy.

function can be further captured through a top-down refinement. A context box located at top right corner shows the function where it belongs in the hierarchy. The PMS components may be added as supporting mechanisms of a function with gray arrows.

3.2. Controllable model construction

To control the manufacturing functions, the functional blocks are added with information flows; thereby, the functional hierarchy is transformed into a controllable hierarchy [13] termed as controllable model:

$$\Sigma \mathrm{CM} = \{ \mathrm{CM}_i(\mathrm{C,I,In,Out}) | i \text{ is an index} \}$$

where C is a set of controllable blocks; I is a set of information flows; both input function In: $C \rightarrow I$ and output function Out: $C \rightarrow I$ specify the graphic relations between C and I.

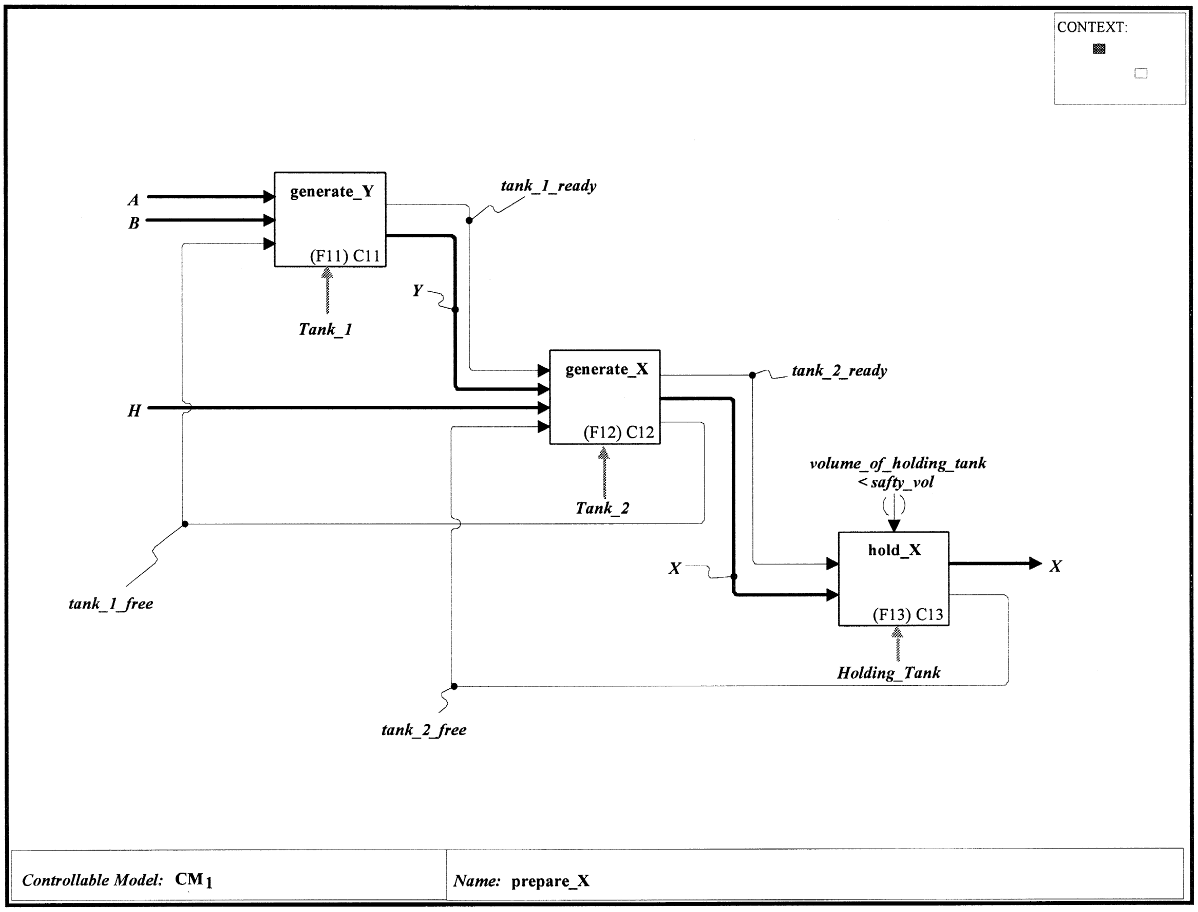


Fig. 6. Controllable function prepare_X (CM_1) in controllable hierarchy.

For instance, CM_1 (Fig. 6) is constructed by adding information flows drawn as thin arrows for coordinating three functions in FM_1 (Fig. 4). A constraint I_{1C13}: *volume_of_Holding_Tank* < *safety_vol* is added to F13 for testing the safety limit of the volume in *Holding Tank*. The representation of CM_1 is as follows:

Model:	CM_1(C,I,In,Out)
C:	{C11,C12,C13}
I:	{tank_1_ready, tank_1_free, tank_2_ready, tank_2_free, I_{1C13}}
In:	In(C11) = {tank_1_free}
	In(C12) = {tank_1_ready, tank_2_free}
	In(C13) = {tank_2_ready, I_{1C13}}
Out:	Out(C11) = {tank_1_ready}
	Out(C12) = {tank_1_free, tank_2_ready}
	Out(C13) = {tank_2_free}

In a controllable model, information flows are designed to control the material flows and manufacturing functions. Two kinds of information flows are constraint flows and control flows. A constraint flow is used to test a specified condition of a functional block, and a control flow is used to either inform the readiness of a

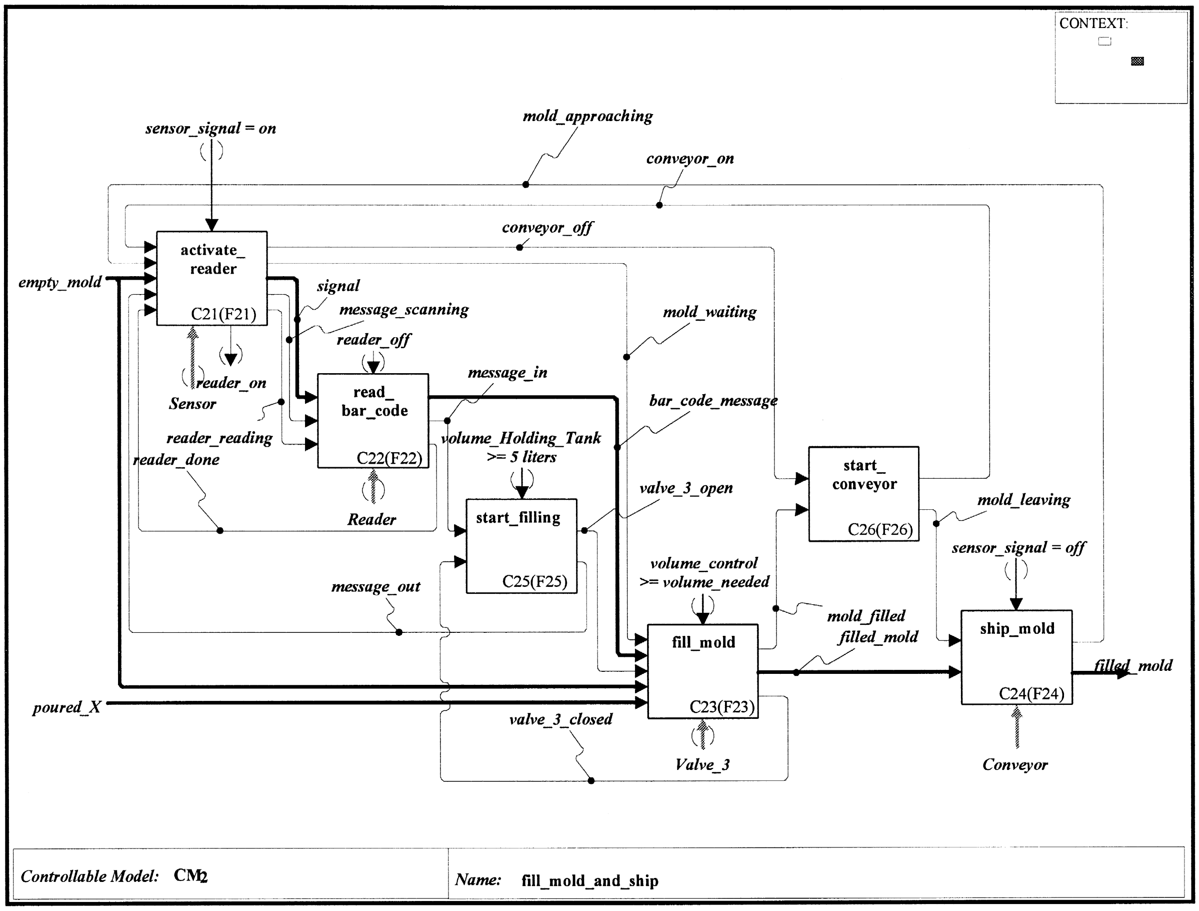


Fig. 7. Controllable function fill_mold_and ship (CM_2) in controllable hierarchy.

material flow or reactivate a manufacturing function. The former is located at the top of a block, and the later is located between two blocks. In CM_1 (Fig. 6), I_{1C13} is a constraint flow, and the others are control flows. The details of each controllable block may be further refined. The terminal blocks in the controllable hierarchy are controllable events which can be fired by information flows.

Similarly, CM_2 (Fig. 7) in the controllable hierarchy corresponds to FM_2 (Fig. 5). When *Sensor* is on, an empty arriving mold on *Conveyor* is detected. C21 turns *Conveyor* off and *Reader* on to scan the barcode on the mold. Once the barcode is read, C22 receives an *off* signal from *Reader*. Next, the barcode message: *volume_needed* is stored into memory. An added event C25 starts the filling process. After the completion of mold filling C23, another added event C26 turns *Conveyor* on to ship out the mold by C24. When the filled mold has not been detected by *Sensor*, the entire process is repeated.

3.3. Dynamic model construction

A dynamic model is constructed to observe and then verify the controllable model's dynamic behavior. The model is represented by a hierarchy of Petri nets [15,16] which is transformed from the controllable hierarchy by adding places and tokens. Tokens may occur concurrently to indicate the dynamics of information flows. A

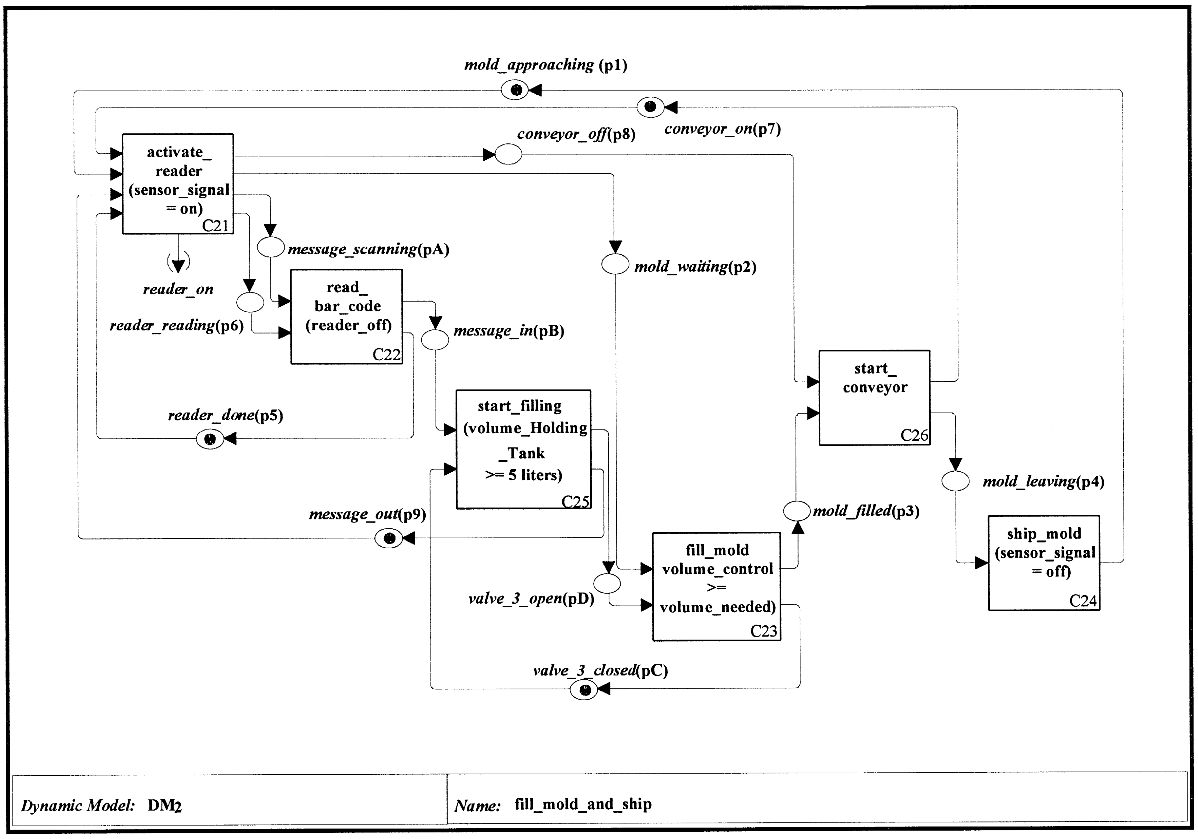


Fig. 8. Petri net fill_mold_and_ship (DM_2) in dynamic hierarchy.

block with only control flows is operationally equivalent to a transition in Petri net, and a block involving constraint flows is equivalent to an externally constrained transition [17]. The model is represented as

$$\Sigma \mathrm{DM} = \{\mathrm{DM}_i(\mathrm{T},\mathrm{P},\mathrm{In},\mathrm{Out},\mu) \mid i \text{ is an index}\}$$

where T is a set of transitions; P is a set of places; functions In:T → P and Out:T → P specify the graphic relations between T and P; and μ represents its initial marking.

A Petri net DM_2 (Fig. 8) is transformed from CM_2 *fill_mold_and_ship* (Fig. 7), and its representation is:

Model:	DM_2(T, P, In, Out, μ)
T:	{C21, C22, C25, C23, C26, C24}
P:	{p1, p2, p3, p4, p5, p6, p7, p8, p9, pA, pB, pC, pD}
In:	I(C21) = {p1, p5, p7, p9}
	I(C22) = {p6, pA}
	I(C25) = {pB, pC}
	I(C23) = {p2, pD}
	I(C26) = {p3, p8}
	I(C24) = {p4}

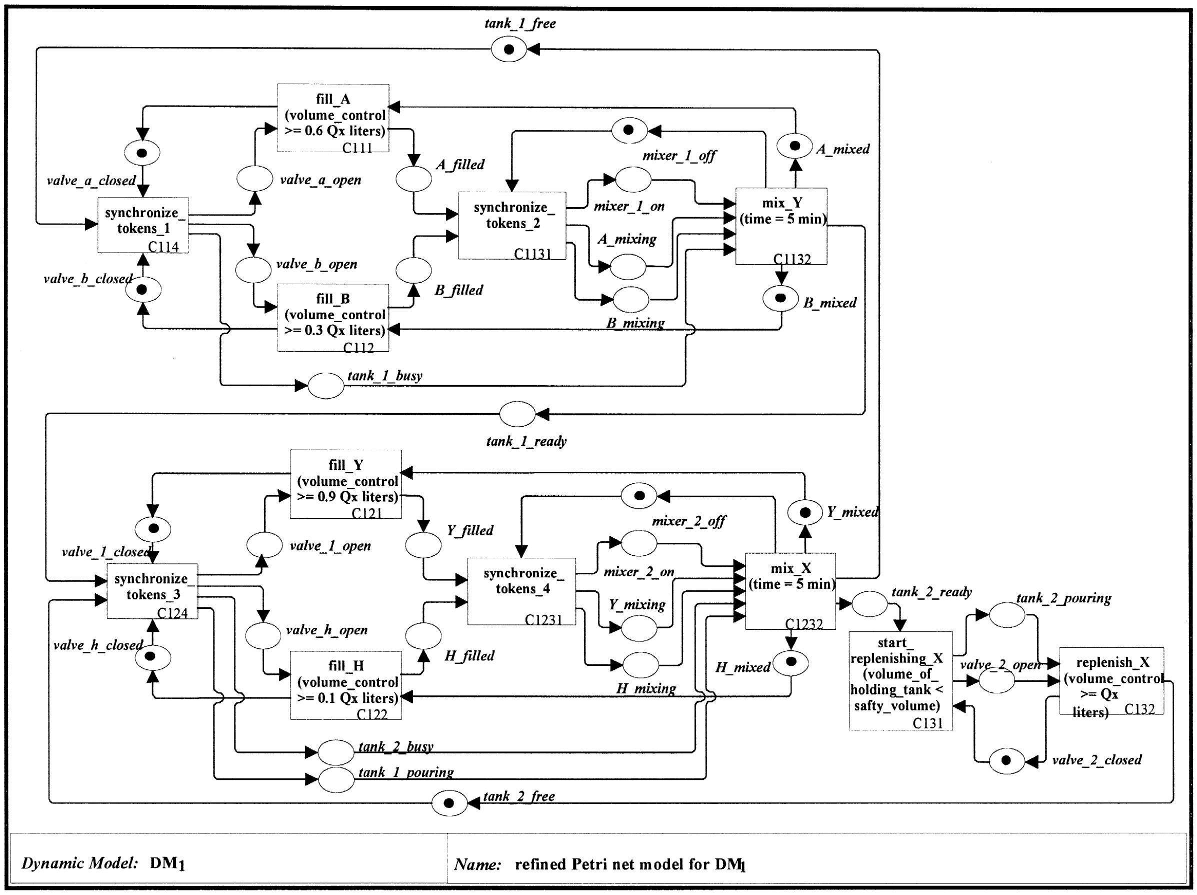

tank_1_free
fill_A (volume_control >= 0.6 Qx liters) C111
valve_a_closed
valve_a_open
A_filled
mixer_1_off
A_mixed
synchronize_ tokens_1 C114
synchronize_ tokens_2 C1131
mixer_1_on
mix_Y (time = 5 min) C1132
A_mixing
valve_b_open
valve_b_closed
fill_B (volume_control >= 0.3 Qx liters) C112
B_filled
B_mixed
B_mixing
tank_1_busy
tank_1_ready
fill_Y (volume_control >= 0.9 Qx liters) C121
Y_filled
mixer_2_off
Y_mixed
valve_1_open
valve_1_closed
synchronize_ tokens_3 C124
synchronize_ tokens_4 C1231
mixer_2_on
mix_X (time = 5 min) C1232
Y_mixing
tank_2_ready
tank_2_pouring
valve_h_open
valve_h_closed
fill_H (volume_control >= 0.1 Qx liters) C122
H_filled
H_mixed
H_mixing
start_ replenishing_X (volume_of_ holding_tank < safty_volume) C131
valve_2_open
replenish_X (volume_control >= Qx liters) C132
valve_2_closed
tank_2_busy
tank_1_pouring
tank_2_free
Dynamic Model: DM1
Name: refined Petri net model for DM1

Fig. 9. The refined Petri net of prepare_X (DM_1).

Out: O(C21) = {p2, p6, p8, pA}
O(C22) = {p5, pB}
O(C25) = {p9, pD}
O(C23) = {p3, pC}
O(C26) = {p4, p7}
μ: O(C24) = {p1}μ:[1 0 0 0 1 0 1 0 1 0 0 1 0]

Similarly, the details of dynamics in DM_1 are synthesized into a large Petri net (Fig. 9). After the construction of DM_1 and DM_2, simulation is performed to verify the dynamic behavior.

Next, the Petri nets are used to define VDs' attributes. Each VD may possess control-related and/or constraint-related attributes. In a given net, control-related attributes can be found from attribute equations [12]. An attribute equation satisfies $\sum \beta_i N(p_i) = 1$, $\beta_i = 1$ or 0, where $N(p_i)$ is the number of tokens in place p_i. In a Petri net, only one place is active in an attribute equation. Thus, each attribute equation may represent one attribute of a VD. Those attribute equations minimally covering all places of a net, called minimal cover, can be obtained through computation [12]. Table 1 lists the attribute equations of an obtained minimal cover of DM_2.

Constraint-related attributes are specified from constraint flows by their physical meanings. For instance, the

Table 1
A minimal cover of DM_2

Attribute equation	Places	Control-related attribute of VD
$N(p1)+N(p2)+N(p3)+N(p4)=1$	p1, p2, p3, p4	Status of Mold
$N(p5)+N(p6)=1$	p5, p6	Status of Reader
$N(p7)+N(p8)=1$	p7, p8	Status of Conveyor
$N(p9)+N(pA)+N(pB)=1$	p9, pA, pB	Message of Reader
$N(pC)+N(pD)=1$	pC, pD	Status of Valve 3

constraint-related attribute *volume_control* of *Valve 3* is specified from the constraint of C23 in Fig. 8: *volume_control* $>=$ *volume_needed*.

After obtaining the attributes of DM_2, a bill of virtual devices (BOD) is formed and their attributes are listed in Table 2. The column labeled ‘type’ tells how to obtain each attribute: from attribute equation (AE) or from constraint (CS) and the column labeled ‘role’ tells the characteristic of each attribute: output, input, or state.

Our computed VDs are different from those objects specified in an object-oriented approach such as in [14]. Objects in that approach are subjectively specified by experienced designers. In our approach, objects are computed through the attribute equations and minimal cover.

3.4. Commanding model construction

A commanding model can be constructed by treating each transition as a rule. Input places of a transition and constraints are antecedents of a rule, and output places are consequences of the rule. Thus the model is represented by a commanding hierarchy as

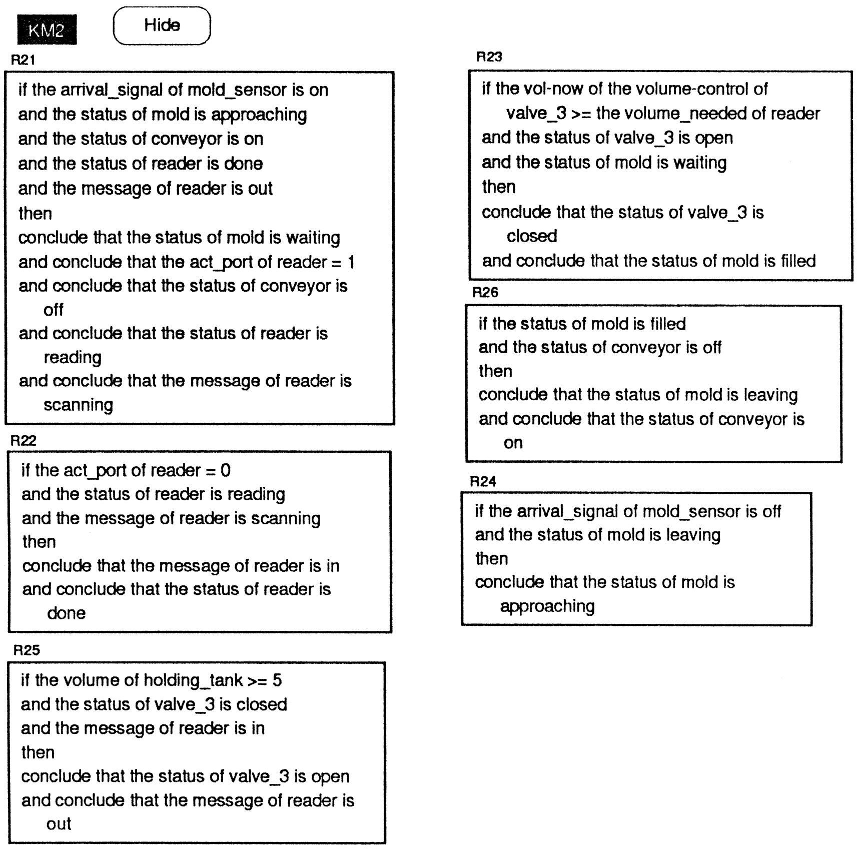
$$\Sigma KM = \{KM_i(R) | i \text{ is an index}\}$$

where R is a set of rules.

After the commanding model is constructed, sets of rules are designed. Each set represents a supervisor's control logic. For instance, the commanding model of *Mold Filling System* possesses two sets of rules: KM_1 for DM_1 *prepare_X* (Fig. 9) and KM_2 for DM_2 *fill_and_ship_mold* (Fig. 8.) Thus, two supervisors are designed.

Table 2
Bill of virtual devices in DM_2

Virtual device	Attribute	Range	Type	Role
Holding_Tank	volume	0–400 L	CS	Input
Valve_3	status	Open, closed	AE	Output
	volume_control	0–5 L	CS	Input
Reader	act_port	On, off	CS	I/O
	status	Done, reading	AE	State
	message	In, out	AE	State
	volume_needed	1–5 L	CS	Input
Mold	status	Approaching, waiting, filled, leaving	AE	State
Sensor	arrival_signal	On, off	CS	Input
Conveyor	status	On, off	AE	Output



R21

```
if the arrival_signal of mold_sensor is on
and the status of mold is approaching
and the status of conveyor is on
and the status of reader is done
and the message of reader is out
then
conclude that the status of mold is waiting
and conclude that the act_port of reader = 1
and conclude that the status of conveyor is
    off
and conclude that the status of reader is
    reading
and conclude that the message of reader is
    scanning
```

R22

```
if the act_port of reader = 0
and the status of reader is reading
and the message of reader is scanning
then
conclude that the message of reader is in
and conclude that the status of reader is
    done
```

R25

```
if the volume of holding_tank >= 5
and the status of valve_3 is closed
and the message of reader is in
then
conclude that the status of valve_3 is open
and conclude that the message of reader is
    out
```

R23

```
if the vol-now of the volume-control of
    valve_3 >= the volume_needed of reader
and the status of valve_3 is open
and the status of mold is waiting
then
conclude that the status of valve_3 is
    closed
and conclude that the status of mold is filled
```

R26

```
if the status of mold is filled
and the status of conveyor is off
then
conclude that the status of mold is leaving
and conclude that the status of conveyor is
    on
```

R24

```
if the arrival_signal of mold_sensor is off
and the status of mold is leaving
then
conclude that the status of mold is
    approaching
```

Fig. 10. Rules of Supervisor KM_2.

Fig. 10 presents the rules of supervisor KM_2, and those rules are transformed from Fig. 8. The rule syntax follows the grammar of G2 [18] which is a real time expert system.

4. Monitor design

A monitor may utilize the system dynamics determined from Petri nets to detect symptoms. The design includes the following steps:

1. list all worlds and legal worlds;
2. specify first principles [19] to justify PMS behavior; and
3. specify symptom detecting rules for each legal world.

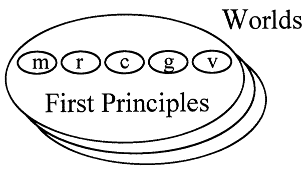


Fig. 11. Worlds of DM_2.

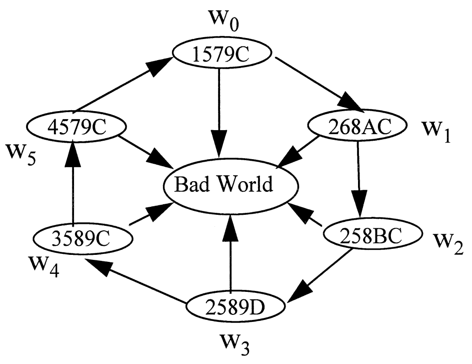
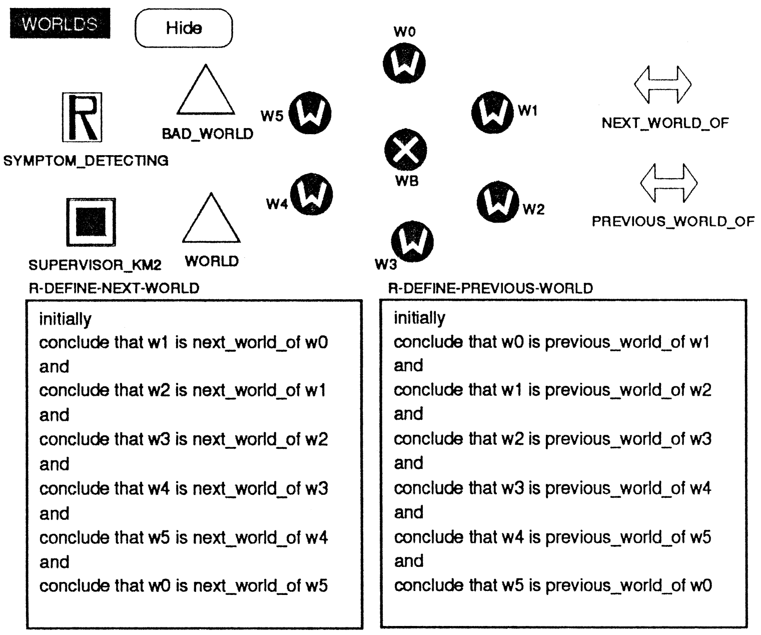


Fig. 12. System dynamics of DM_2.

4.1. Worlds

All worlds are generated from the possible combination of tokens in control-related attributes of VDs. For instance, Fig. 11 shows the worlds of DM_2, where control-related attributes can be found in Table 1. All worlds are represented as w($mrcgv$), where m is the status of mold: $m = 1, 2, 3,$ or 4; r is the status of *Reader*: $r = 5$ or 6; c is the status of *Conveyor*: $c = 7$ or 8; g is the message of *Reader*: $g = 9$, A, or B; and v is the status of *Valve 3*: $v =$ C or D. Here the hexadecimal number represents place code in DM_2. All legal worlds are obtained from the reachable marking of DM_2 as illustrated in Fig. 12, and they are w_0(1579C): mold approaching, w_1(268AC): reading barcode, w_2(258BC): waiting to fill, w_3(2589D): filling mold, w_4(3589C): mold filled, and w_5(4579C) mold leaving, where the subscript denotes world code. Accordingly, a world-based monitor is designed as shown in Fig. 13, where all legal worlds and transitions are defined. The current world transits into either its next legal world or a bad world (WB).



R-DEFINE-NEXT-WORLD

```
initially
conclude that w1 is next_world_of w0
and
conclude that w2 is next_world_of w1
and
conclude that w3 is next_world_of w2
and
conclude that w4 is next_world_of w3
and
conclude that w5 is next_world_of w4
and
conclude that w0 is next_world_of w5
```

R-DEFINE-PREVIOUS-WORLD

```
initially
conclude that w0 is previous_world_of w1
and
conclude that w1 is previous_world_of w2
and
conclude that w2 is previous_world_of w3
and
conclude that w3 is previous_world_of w4
and
conclude that w4 is previous_world_of w5
and
conclude that w5 is previous_world_of w0
```

Fig. 13. A world-based monitor for DM_2.

4.2. First principles

In each legal world, PDs or controllers should follow the world's first principles [19]. The principles may be specified in qualitative, in quantitative, or in temporal forms. For instance, in world w_3 (filling mold), a qualitative principle states that the filled volume should be increased. At this moment, if no flow is indicated, then the behavior of *Valve 3* is against the principle. Also in world w_3, a quantitative principle states that the filled volume should be near the expected value having been computed from a mathematical model. In world w_2 (reading barcode), a temporal principle states that the reading should be completed in 10 s according to the reader's specification. The first two types of first principles generally require sensory feedback information, while the last only needs a watchdog timer.

4.3. Symptom detection

In previous literature, symptoms have been detected from qualitative physics [20], process equation [5,6], maximum token holding time [21], and token conservation [22]. To equip a monitor with at least the capability to detect those symptoms, four types of rules are designed: qualitative rules, quantitative rules, time-out rules, and next legal world rules. The first three types of rules utilize qualitative, quantitative, and temporal principles, respectively, to detect symptoms in PMS, while the last type utilizes system dynamics to detect wrong world transitions. Fig. 14 presents the symptom detecting rules for DM_2.

For instance, a qualitative rule named *DETECTING_V3_NO_FLOW* detects no flow of *Valve 3* in world w_3 and informs the operator an alarm message. A quantitative rule named *DETECTING_V3_INACCURACY* detects abnormal flow volume from *Valve 3*. A mathematical model implemented as a simulator provides the expected value (*vol-simulated*). The observed value (*vol-now*) is compared with the expected one. When a discrepancy over a specified threshold arises, *Valve 3* incurs an accuracy problem.

A generic time-out rule named *CHECKING_TIME-OUT* detects time-out in the current world. When a time-out error occurs, the monitor deactivates *fill_mold_rules* (supervisor KM_2) and activates a troubleshooter for DM_2.

A rule named *CHECKING_W* detects transition occurrences and deactivates KM_2 for checking the new world. If the new world passes through rule *CHECKED_NEXT_W_OK*, the transition is legal and KM_2 is reactivated. Otherwise, the wrong transition will be detected by rule *CHECKED_BAD_W* and troubleshooter rules will be invoked. Normal world transition should follow the system dynamics shown in Fig. 12.

Symptoms detected by qualitative or quantitative rules, e.g., the leakage from a valve stuck at open or a discrepancy over threshold, may provide trivial diagnosis [23] which implies that the fault source is immediately located. The other symptom requires a troubleshooter to locate the faults.

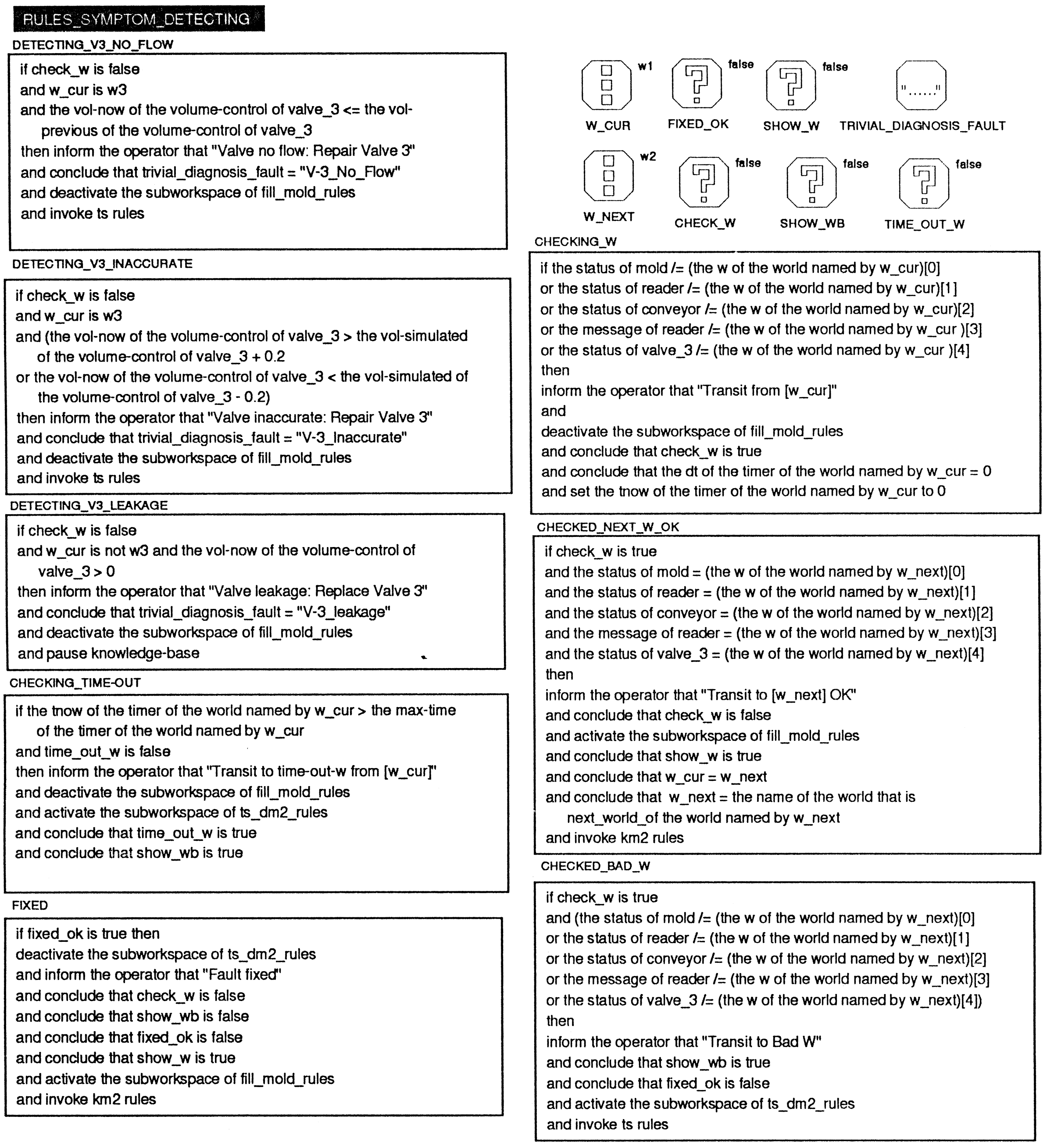
5. Troubleshooter design

A troubleshooter is designed to locate faults (diagnosis) and recover the faults. For such a troubleshooter, five steps are involved:

1. establish a fault-symptom table;
2. analyze the possible next worlds;
3. locate the most probable fault;
4. design auto-recovery plans; and
5. design aid-recovery plans.

For diagnosis (steps 1–3), we use both deep knowledge and shallow knowledge [7] to locate faults. By using deep knowledge, faults can be located through matching detected symptoms with physically fault-caused symptoms at step 1 and/or through the possible next worlds at step 2. By using shallow knowledge, the most probable fault can be determined from maintenance history.

For recovery (steps 4 and 5), auto-recovery plans solve the recoverable faults caused from supervisors as well as related to work-in-process (WIP); aid-recovery plans solve the faults in PMS.



RULES_SYMPTOM_DETECTING

DETECTING_V3_NO_FLOW
```
if check_w is false
and w_cur is w3
and the vol-now of the volume-control of valve_3 <= the vol-
   previous of the volume-control of valve_3
then inform the operator that "Valve no flow: Repair Valve 3"
and conclude that trivial_diagnosis_fault = "V-3_No_Flow"
and deactivate the subworkspace of fill_mold_rules
and invoke ts rules
```

DETECTING_V3_INACCURATE
```
if check_w is false
and w_cur is w3
and (the vol-now of the volume-control of valve_3 > the vol-simulated
   of the volume-control of valve_3 + 0.2
or the vol-now of the volume-control of valve_3 < the vol-simulated of
   the volume-control of valve_3 - 0.2)
then inform the operator that "Valve inaccurate: Repair Valve 3"
and conclude that trivial_diagnosis_fault = "V-3_Inaccurate"
and deactivate the subworkspace of fill_mold_rules
and invoke ts rules
```

DETECTING_V3_LEAKAGE
```
if check_w is false
and w_cur is not w3 and the vol-now of the volume-control of
   valve_3 > 0
then inform the operator that "Valve leakage: Replace Valve 3"
and conclude that trivial_diagnosis_fault = "V-3_leakage"
and deactivate the subworkspace of fill_mold_rules
and pause knowledge-base
```

CHECKING_TIME-OUT
```
if the tnow of the timer of the world named by w_cur > the max-time
   of the timer of the world named by w_cur
and time_out_w is false
then inform the operator that "Transit to time-out-w from [w_cur]"
and deactivate the subworkspace of fill_mold_rules
and activate the subworkspace of ts_dm2_rules
and conclude that time_out_w is true
and conclude that show_wb is true
```

FIXED
```
if fixed_ok is true then
deactivate the subworkspace of ts_dm2_rules
and inform the operator that "Fault fixed"
and conclude that check_w is false
and conclude that show_wb is false
and conclude that fixed_ok is false
and conclude that show_w is true
and activate the subworkspace of fill_mold_rules
and invoke km2 rules
```

CHECKING_W
```
if the status of mold /= (the w of the world named by w_cur)[0]
or the status of reader /= (the w of the world named by w_cur)[1]
or the status of conveyor /= (the w of the world named by w_cur)[2]
or the message of reader /= (the w of the world named by w_cur )[3]
or the status of valve_3 /= (the w of the world named by w_cur )[4]
then
inform the operator that "Transit from [w_cur]"
and
deactivate the subworkspace of fill_mold_rules
and conclude that check_w is true
and conclude that the dt of the timer of the world named by w_cur = 0
and set the tnow of the timer of the world named by w_cur to 0
```

CHECKED_NEXT_W_OK
```
if check_w is true
and the status of mold = (the w of the world named by w_next)[0]
and the status of reader = (the w of the world named by w_next)[1]
and the status of conveyor = (the w of the world named by w_next)[2]
and the message of reader = (the w of the world named by w_next)[3]
and the status of valve_3 = (the w of the world named by w_next)[4]
then
inform the operator that "Transit to [w_next] OK"
and conclude that check_w is false
and activate the subworkspace of fill_mold_rules
and conclude that show_w is true
and conclude that w_cur = w_next
and conclude that  w_next = the name of the world that is
   next_world_of the world named by w_next
and invoke km2 rules
```

CHECKED_BAD_W
```
if check_w is true
and (the status of mold /= (the w of the world named by w_next)[0]
or the status of reader /= (the w of the world named by w_next)[1]
or the status of conveyor /= (the w of the world named by w_next)[2]
or the message of reader /= (the w of the world named by w_next)[3]
or the status of valve_3 /= (the w of the world named by w_next)[4])
then
inform the operator that "Transit to Bad W"
and conclude that show_wb is true
and conclude that fixed_ok is false
and activate the subworkspace of ts_dm2_rules
and invoke ts rules
```

Fig. 14. Symptom detecting rules for DM_2.

Table 3
The fault-symptom table of DM_2

Fault source	Code	w_0(1579C)	w_1(268AC)	w_2(258BC)	w_3(2589D)	w_4(3589C)	w_5(4579C)
PMS:							
Conveyor stuck	f_1	Time-out					Time-out
Sensor bad always on	f_2	Error to w_1	Time-out				Time-out
Sensor bad always off	f_3	Time-out					Error to w_0
Reader bad	f_4		Time-out				
Valve stuck at closed	f_5				1. Against laws of physics 2. Time-out		
Valve stuck at open	f_6	X leakage	X leakage	X leakage		X leakage	X leakage
WIP:							
No mold	f_7	Time-out					
Bar code problem	f_8		Time-out				
Insufficient X in Holding Tank	f_9			Time-out			

Table 4
The possible next worlds of DM_2

Current legal world (*mrcgv*)	Next world (*mrcgv*)	Classified zone	Operational status	Auto-recovery plan and message to operator
w_0(1579C) *Mold approaching*	w_1(268AC)	Legal	Normal	168AC → 268AC
	w(168AC)	One-token-error	Abnormal mold status	
	w(2589C)	One-token-error	Abnormal reader status and message	2589C → 268AC
	w(267AC)	one-token-error	Abnormal conveyor status	267AC → 268AC
	w_0(1579C)	time-out	Possible faults f_1, f_3, f_7	Warning
w_1(268AC) *Reading barcode*	w_2(258BC)	Legal	Normal	
	w_1(268AC)	Time-out	Possible faults f_2, f_4, f_8	Reset and re-read warning
w_2(258BC) *Waiting to fill*	w_3(2589D)	Legal	Normal	
	w(258BD)	One-token-error	Abnormal reader message	258BD → 2589D
	w(2589C)	One-token-error	Abnormal valve status	System suspending
	w_2(258BC)	Time-out	Possible fault f_9	Warning
w_3(2589D) *Filling mold*	w_4(3589C)	Legal	Normal	
	w(2589C)	One-token-error	Abnormal mold status	2589 → 3589C
	w(3589D)	One-token-error	Abnormal valve status	Force valve closed: 3589D → 3589C warning
	w_3(2589D)	Time-out	Possible fault f_5	Force valve closed: 2589D → 2589C system halt
w_4(3589C) *Mold filled*	w_5(4579C)	Legal	Normal	
	W(3579C)	One-token-error	Abnormal mold status	3579C → 4579C
	w(4589C)	One-token-error	Abnormal conveyor status	4589C → 4579C
	w_4(3589C)	Time-out	Abnormal	Force conveyor on: 3589C → 4579C warning
w_5(4579C) *Mold leaving*	w_0(1579C)	Legal	Normal	
	w_5(4579C)	Time-out	Possible faults f_1, f_2	System suspending

5.1. Fault-symptom table

A fault-symptom table lists all physical faults and their caused symptoms in every world. In the table, all probable faults related to PMS and WIP are considered, and the symptoms caused by these faults are analyzed for every legal world. For instance, Table 3 is a fault-symptom table for DM_2. The analyzed symptoms caused by two faults are described here. By assuming that the sensor is bad with *on* signal, two worlds w_0 and w_5 may be affected. If the current world is w_0, then w_0 erroneously transits to w_1 and a time-out error will occur for no mold being read. If the current world is w_5, a time-out error will occur for no *shipping-out* signal being provided by the sensor. By assuming that *Valve 3* is stuck at open, the obvious symptom in worlds w_0, w_1, w_2, w_4, or w_5 is the leakage of material X.

When a symptom is detected in a world, the probable faults are determined. For instance, if a time-out error is detected in w_0, then the probable faults are sensor bad (always on), reader bad, or barcode problem.

5.2. Possible next worlds

Possible next worlds are worlds which may transit from a legal world. These worlds can be analyzed through system dynamics. For instance, Table 4 lists the possible next worlds of DM_2. Following the system dynamics in Fig. 12, the next legal world after w_0(1579C) is w_1(268AC). In a normal situation, four tokens must be updated by the supervisor. However, if one of them is not updated, a one-token error occurs such as w(168AC), w(2589C), and w(267AC). This occurrence implies that the supervisor has failed to update the attribute of a VD in real time. If w_0 has been sustained over a pre-specified time, a time-out error occurs. The error may be caused by f_1, f_3, and f_7 (see Table 3 for reasons). Except for the next legal, one-token error, and time-out worlds, the other worlds are treated as untraceable worlds including those worlds where two or more tokens have not been updated. Correspondingly, the other legal worlds in Table 4 are analyzed.

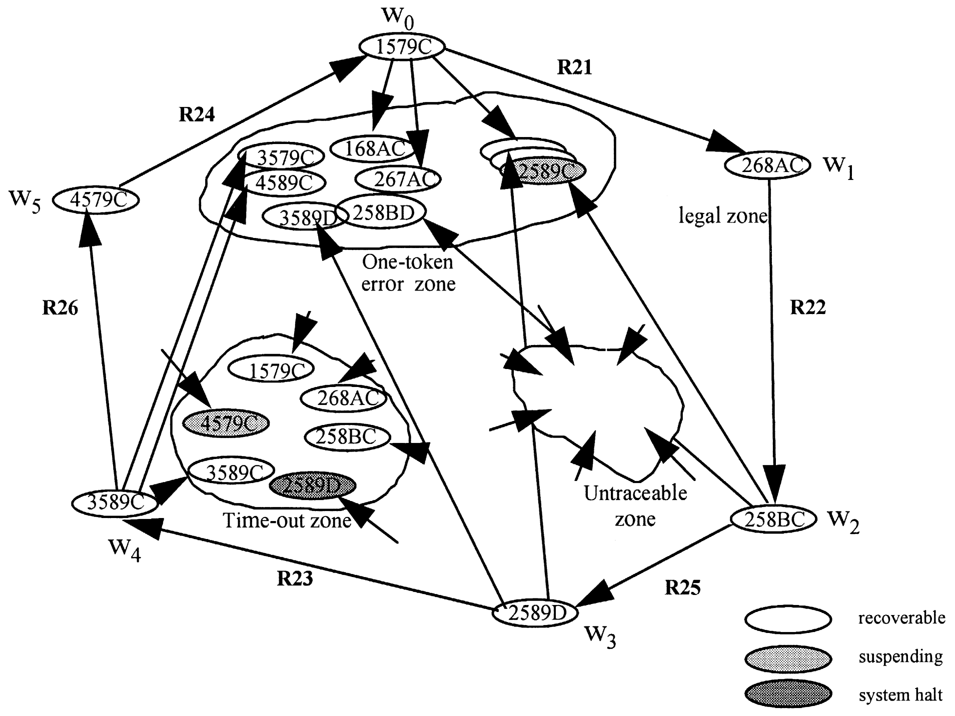


Fig. 15. Worlds in classified zones.

All worlds of DM_2 can be classified into four zones as shown in Fig. 15 including legal, one-token error, time-out, and untraceable zones. Worlds in legal zone follow system dynamics, and they are filtered out by monitors. Worlds in one-token error zone are caused by 'loss-updating' problems. Worlds in time-out zone are worlds whose dwelling time exceeds a specified value. Worlds in untraceable zone are losing their tracks.

5.3. The most probable fault

A symptom in a world may be caused from one fault source in Table 3, such as time-out in w_2 caused by an insufficient amount of X in *Holding Tank*, or time-out in w_3 caused by *Valve 3* stuck at closed, or the one-token errors in Table 4. Thus the only fault source is immediately located. For multiple fault sources, e.g., time-out in w_0, w_1, or w_5 of Table 3, the most probable one should be located.

In a troubleshooter, the most probable fault is found by applying Bayes' theorem [24] which lies in the inversion Eq. (1).

$$P(H|e) = \frac{P(H)\cdot P(eH)}{P(e)} \tag{1}$$

The formula states that the belief we accord a hypothesis H upon obtaining evidence e can be computed by multiplying our previous belief $P(H)$ by likelihood $P(e|H)$ that e will materialize if H is true, where $P(e)$ is a normalizing constant.

In most diagnosis processes, sources of evidence are incrementally collected. Bayes' theorem facilitates the updating of a belief from new evidence. Let $e_n = e_1, e_2, \ldots, e_n$ denote a sequence of data observed in the past, and e denote a new fact. The incremental belief updating can be reformulated as Eq. (2).

$$P(H|e_n,e) = \frac{P(H|e_n)\cdot P(e|e_n,H)}{P(e|e_n)} \tag{2}$$

In Eq. (2), $P(H|e_n)$ is a summarized belief from previous observations, and $P(e|e_n, H)$ is the likelihood which measures the probability of new evidence e, given the hypothesis and the previous observations. For those cases in which likelihood is independent of previous observations and involves only e and H, then

$$P(e|e_n,H) = P(e|H) \tag{3}$$

Eq. (2) can be rewritten as

$$P(H|e_n,e) = \alpha P(H|e_n)\cdot P(e|H) \tag{4}$$

where α is a normalizing constant. An example is given below.

In world w_1 of DM_2, let hypothesis $\boldsymbol{H} = (H_1, H_2, H_3)$ denote time-out faults, where H_1, H_2, and H_3 represent faults caused by the sensor, reader, and barcode, respectively (see Table 3 for reasons). The original belief or prior probability $P(\boldsymbol{H})$ regarding faults is given by system designers, and $P(\boldsymbol{H}) = [0.15, 0.10, 0.75]$. Based on the shallow knowledge from maintenance history, the likelihood $P(e_j|H_i)$ with respect to bodies of evidence has been collected from e_1: observation of mold existence, e_2: examination the on/off light of *Reader*, and e_3: observation of laser scanning beam, when a time-out error occurs in w_1. Assume that the collected likelihood ($\sim$ means negation) are listed as Table 5. If evidence e_1, e_2, e_3 are incrementally collected after time-out, according to Eq. (4)

$$P(H|e_1) = \alpha_1\cdot[0.15, 0.10, 0.75]\cdot[0.10, 0.95, 1.00] = [0.017, 0.110, 0.872]$$

$$P(H|e_1,e_2) = \alpha_2[0.017, 0.110, 0.872]\cdot[0.98, 0.10, 0.95] = [0.019, 0.013, 0.968]$$

$$P(H|e_1,e_2,e_3) = \alpha_3\cdot[0.019, 0.013, 0.968]\cdot[0.95, 0.05, 0.99] = [0.019, 0.001, 0.980]$$

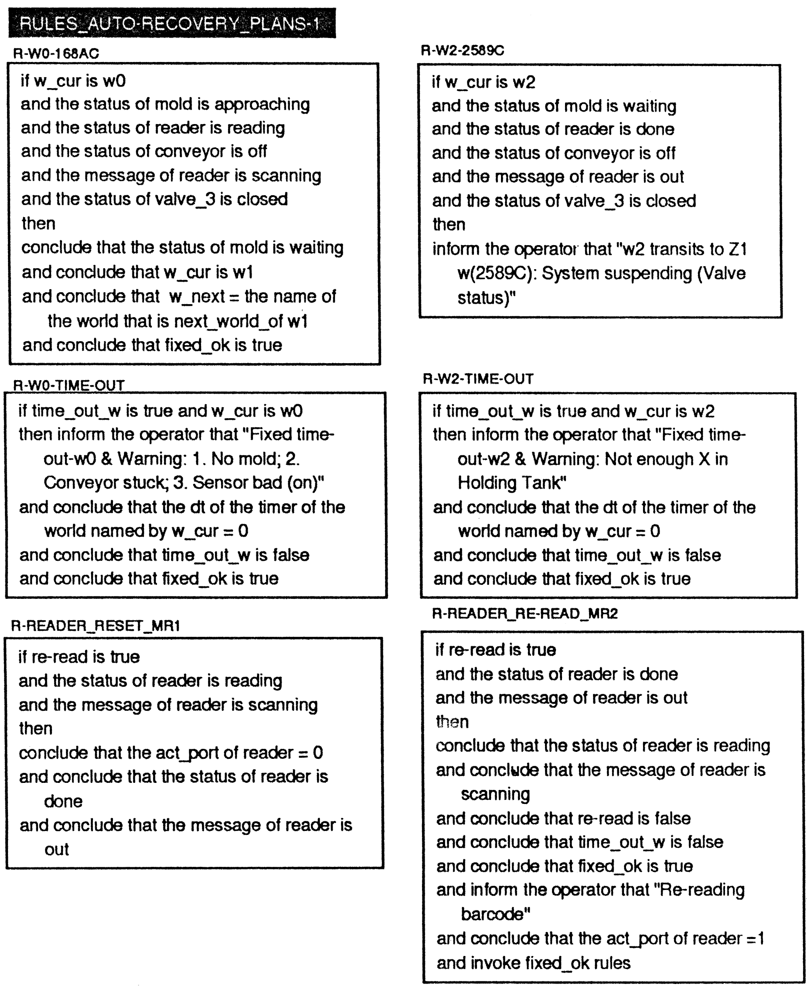
Thus the most probable fault is barcode problem.

Table 5
The collected likelihood for the time-out of barcode reading

Time-out in w_1 $P(e_j \mid H_i)$	e_1 Mold existing	$e_{1'}$ ~ Mold existing	e_2 Read light on	$e_{2'}$ Read light off	e_3 Laser beam OK	$e_{3'}$ ~ Laser beam OK
H_1: sensor	0.10	0.90	0.98	0.02	0.95	0.05
H_2: reader	0.95	0.05	0.10	0.90	0.05	0.95
H_3: barcode	1.00	0.00	0.95	0.05	0.99	0.01

5.4. Auto-recovery plans

Auto-recovery plans are designed to automatically solve the supervisory faults (one-token errors) and WIP problems. For a one-token error, an auto-recovery plan updates the error token by a token-updating rule. For instance, one-token errors in DM_2 are listed in Table 4, and the corresponding token-updating rules are shown in the last column. By applying a token-updating rule listed in Table 4, the bad world w(168AC) transiting from w_0 is rewritten as legal world w_1(268AC). A designed rule named *R-W0-168AC* in Fig. 16 updates the status of


RULES_AUTO-RECOVERY_PLANS-1

R-W0-168AC
```
if w_cur is w0
and the status of mold is approaching
and the status of reader is reading
and the status of conveyor is off
and the message of reader is scanning
and the status of valve_3 is closed
then
conclude that the status of mold is waiting
and conclude that w_cur is w1
and conclude that w_next = the name of
    the world that is next_world_of w1
and conclude that fixed_ok is true
```

R-W2-2589C
```
if w_cur is w2
and the status of mold is waiting
and the status of reader is done
and the status of conveyor is off
and the message of reader is out
and the status of valve_3 is closed
then
inform the operator that "w2 transits to Z1
    w(2589C): System suspending (Valve
    status)"
```

R-W0-TIME-OUT
```
if time_out_w is true and w_cur is w0
then inform the operator that "Fixed time-
    out-w0 & Warning: 1. No mold; 2.
    Conveyor stuck; 3. Sensor bad (on)"
and conclude that the dt of the timer of the
    world named by w_cur = 0
and conclude that time_out_w is false
and conclude that fixed_ok is true
```

R-W2-TIME-OUT
```
if time_out_w is true and w_cur is w2
then inform the operator that "Fixed time-
    out-w2 & Warning: Not enough X in
    Holding Tank"
and conclude that the dt of the timer of the
    world named by w_cur = 0
and conclude that time_out_w is false
and conclude that fixed_ok is true
```

R-READER_RESET_MR1
```
if re-read is true
and the status of reader is reading
and the message of reader is scanning
then
conclude that the act_port of reader = 0
and conclude that the status of reader is
    done
and conclude that the message of reader is
    out
```

R-READER_RE-READ_MR2
```
if re-read is true
and the status of reader is done
and the message of reader is out
then
conclude that the status of reader is reading
and conclude that the message of reader is
    scanning
and conclude that re-read is false
and conclude that time_out_w is false
and conclude that fixed_ok is true
and inform the operator that "Re-reading
    barcode"
and conclude that the act_port of reader = 1
and invoke fixed_ok rules
```

Fig. 16. Some mentioned auto-recovery plans for DM_2.

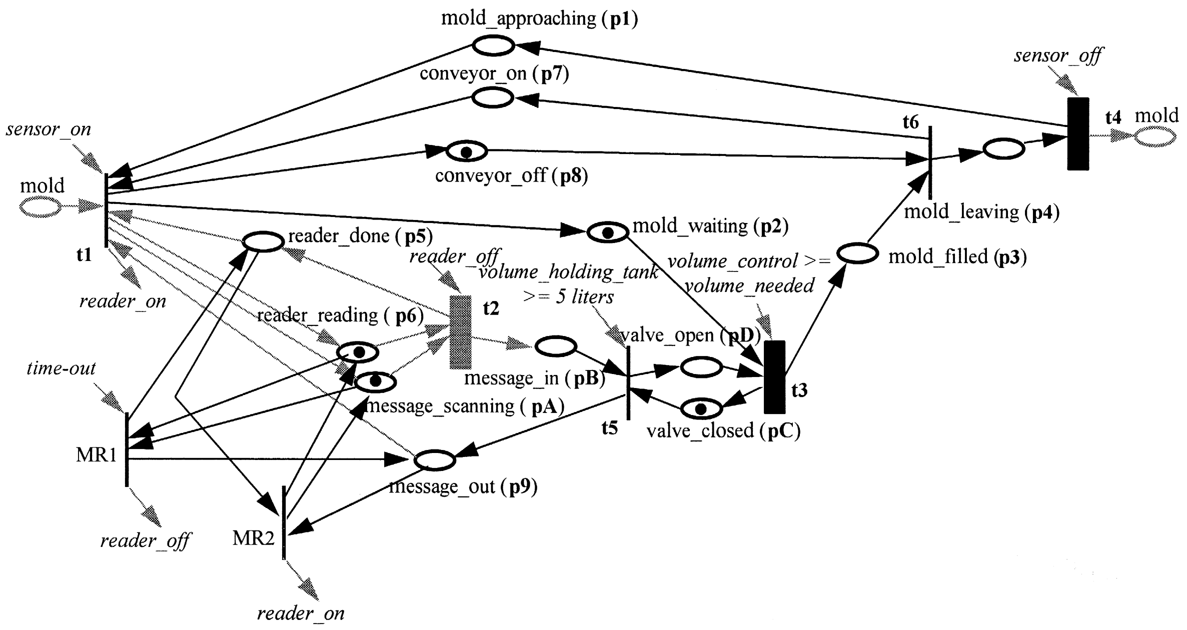


Fig. 17. Maintenance rules to reset Reader (MR1) and re-read (MR2).

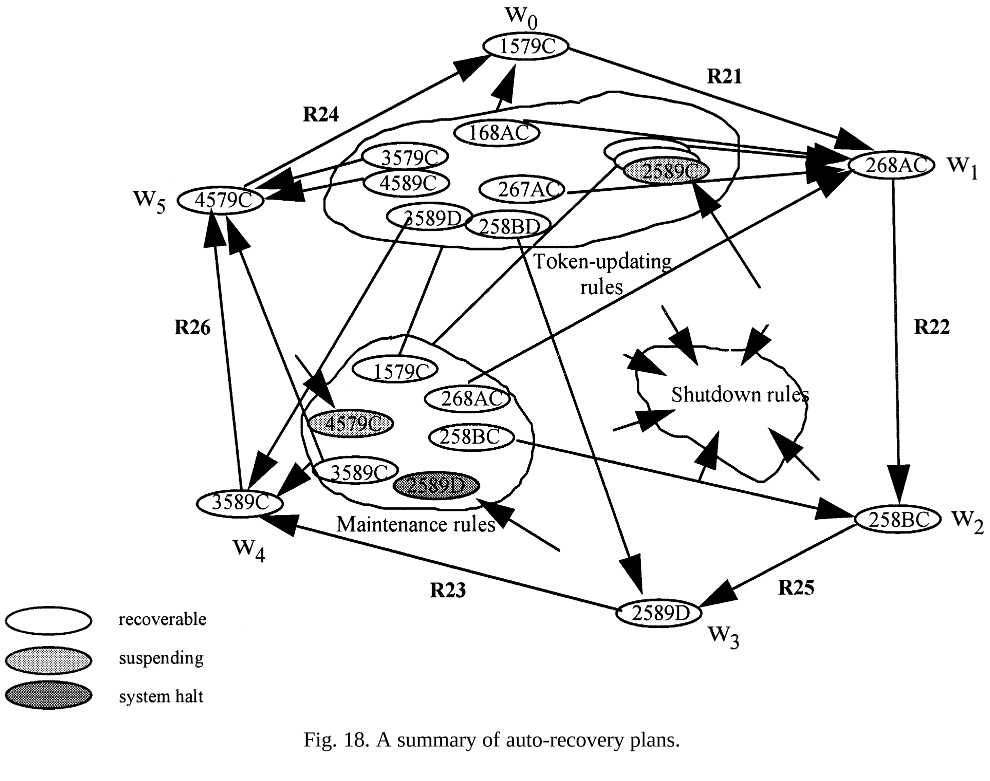


Fig. 18. A summary of auto-recovery plans.

mold from *mold_approaching* to *mold_waiting*. Applying token-updating rules is an efficient way to solve supervisor's token problems. However, those rules should be carefully designed to avoid side-effects, particularly those activating VDs. For instance, in Table 4, world w(2589C) after a transition from w_2 is conservatively considered as a bad world and, thus, no updating rule is applied to avoid erroneously opening *Valve 3*. A rule named *R-W2-2589C* in Fig. 16 only sends a message 'w_2 transits to zone 1 (one-token error): System suspending (abnormal valve status).'

For time-out errors, a troubleshooter reacts with different maintenance rules: warning, retrial (rework), skipping, and shutdown, which covers all the exception handling in [9] and error recovery in [8,10]. If time-out related to WIP does not damage the system after the fault, a warning message is sufficient. For instance, in Table 4, time-out in worlds w_0 and w_2 can be neglected for a period of time, which may be caused from no arrival mold or an insufficient amount of material X in *Holding Tank*, respectively. Even if time-out in w_0 is caused by either a conveyor or a sensor fault, the system will not be damaged any further since no activating command is generated. Two rules, *R-W0-TIME-OUT* and *R-W2-TIME-OUT* in Fig. 16, are designed for warning. However, for time-out in w_3, the troubleshooter forces *Valve 3* closed, informs the operator 'checking Valve 3,' and commands the system to be halted for ensuring the quality of filling process. If a time-out error is caused by an information problem related to WIP such as reading a 'vague' barcode in w_2, a retrial action is executed. Maintenance rules MR1 and MR2 are applied to allow *Reader* to be reset and then to re-read as shown in Fig. 17. Tokens in places p6 and pA fire MR1 when a time-out error occurs; the *act_port* of *Reader* is then reset; and tokens are moved to p5 and p9. Tokens in p5 and p9 subsequently fire MR2 which reactivates *Reader* by setting *act_port* on. These two rules are implemented as *R-READER-RESET-MR1* and *R-READER-RESET-MR1* in Fig. 16. If the second read trial also fails, the troubleshooter skips the mold with the message 'Unfilled mold.' If unfilled molds successively occur, it informs the operator 'Check Reader and Sensor' (see Table 3 for reasons).

Fig. 18 summarizes auto-recovery plans for worlds in one-token error zone and time-out zones. Token-updating rules fix one-token errors. Maintenance rules reset the world clocks of worlds w_0, w_1, and w_2 by retrial, and change time-out world w_4 into w_5. For untraceable worlds (since they are far from being recovered), troubleshooter commands system to be halted by applying shutdown rules.

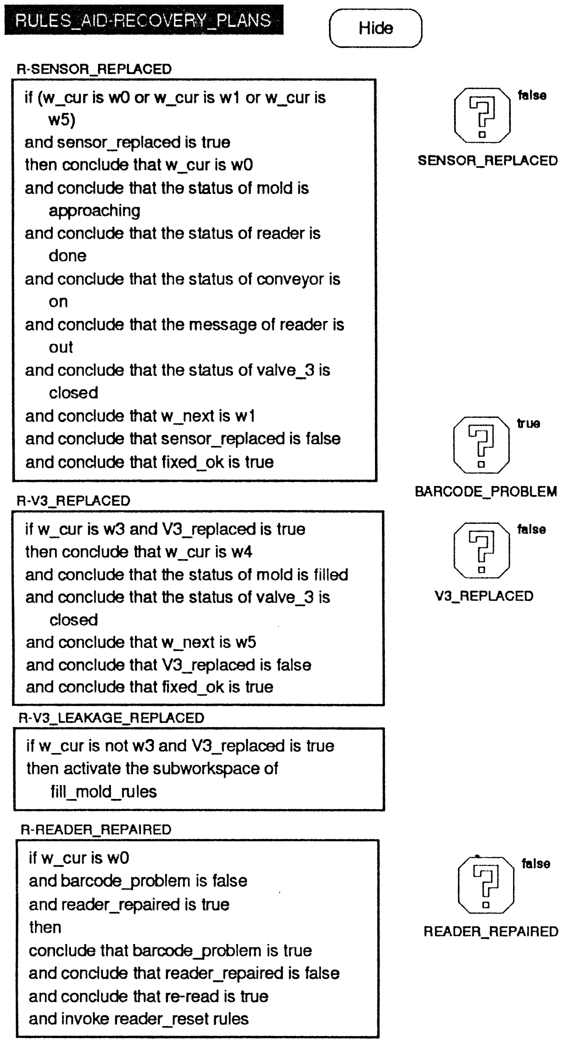
Table 6
Aid-recovery plans for faulty devices (symptoms are shaded)

Fault	w_0(1579C)	w_1(268AC)	w_2(258BC)	w_3(2589D)	w_4(3589C)	w_5(4579C)
f_1: Conveyor stuck	Time-out Repair Conveyor					Time-out Repair Conveyor
f_2: Sensor bad always on	Error to w_1	time-out Replace Sensor change into w_0				Time-out Replace Sensor change into w_0
f_3: Sensor bad always off	Time-out Replace Sensor					Error to w_0
f_4: Reader bad		Time-out Repair Reader reset Reader re-read				
f_5: Valve stuck at closed				1. Against laws of physics 2. Time out replace V3 change into w_4		
f_6: Valve stuck at open	X leakage Replace V3	X leakage Replace V3	X leakage Replace V3		X leakage Replace V3	X leakage Replace V3

5.5. Aid-recovery plans

An aid-recovery plan is only activated after the troubleshooter acknowledges from the operator that a faulty source has been eliminated. Once it acknowledges, an aid-recovery plan also continues the error recovery without human intervention. The design involves changing the current world into an appropriate legal world and adding some control actions.

Table 6 lists the aid-recovery plans for DM_2. Each plan is designed for a fault in Table 3. For instance, no recovery plan after repairing the conveyor is available since the system's state is normal. For a bad sensor (always on), after replacement, the world is changed into w_0. For a bad sensor (always off), sensor replacement is sufficient. For a bad reader, after repairing it, maintenance rules MR1 and MR2 are successively applied to



R-SENSOR_REPLACED

```
if (w_cur is w0 or w_cur is w1 or w_cur is
    w5)
and sensor_replaced is true
then conclude that w_cur is w0
and conclude that the status of mold is
    approaching
and conclude that the status of reader is
    done
and conclude that the status of conveyor is
    on
and conclude that the message of reader is
    out
and conclude that the status of valve_3 is
    closed
and conclude that w_next is w1
and conclude that sensor_replaced is false
and conclude that fixed_ok is true
```

R-V3_REPLACED

```
if w_cur is w3 and V3_replaced is true
then conclude that w_cur is w4
and conclude that the status of mold is filled
and conclude that the status of valve_3 is
    closed
and conclude that w_next is w5
and conclude that V3_replaced is false
and conclude that fixed_ok is true
```

R-V3_LEAKAGE_REPLACED

```
if w_cur is not w3 and V3_replaced is true
then activate the subworkspace of
    fill_mold_rules
```

R-READER_REPAIRED

```
if w_cur is w0
and barcode_problem is false
and reader_repaired is true
then
conclude that barcode_problem is true
and conclude that reader_repaired is false
and conclude that re-read is true
and invoke reader_reset rules
```

Fig. 19. Aid-recovery plans for DM_2.

resetting *Reader* and making *Reader* re-read the barcode. For stuck *Valve 3* in world w_3, after replacement, the unsuccessfully-filled mold will be moved out by changing the world into w_4. For valve leakage in world except w_3, after replacement, the filling process continues. Fig. 19 presents these rules. Rule *R-SENSOR_REPLACED* updates the current world to w_0, when the bad sensor is replaced. Rule *R-V3_REPLACED* changes w_3 into w_4

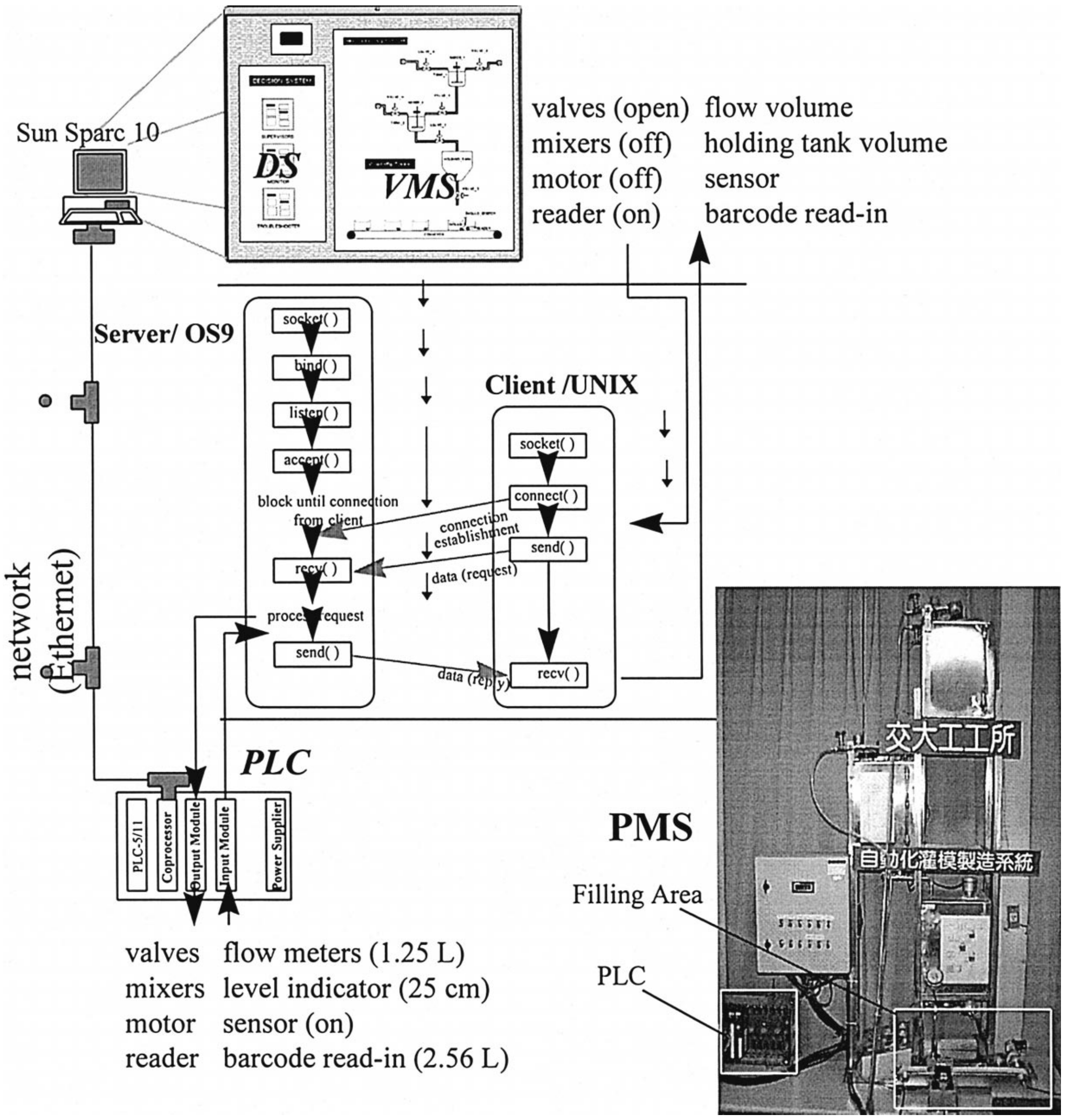


Fig. 20. Configuration of automated Mold Filling System.

for shipping out the unfinished mold. Rule *R-V3_LEAKAGE_REPLACED* recovers the process to normal state. Rule *R-READER_REPAIRED* invokes *R-TIME-OUT-W1*, *MR1*, and *MR2* when *Reader* has been repaired.

6. Implementation

The final stage for building the AMS involves installing PMS and connecting PMS to VMS. According to the design, an automated mold milling system was already installed at Shop Floor Control Lab., NCTU. Fig. 20

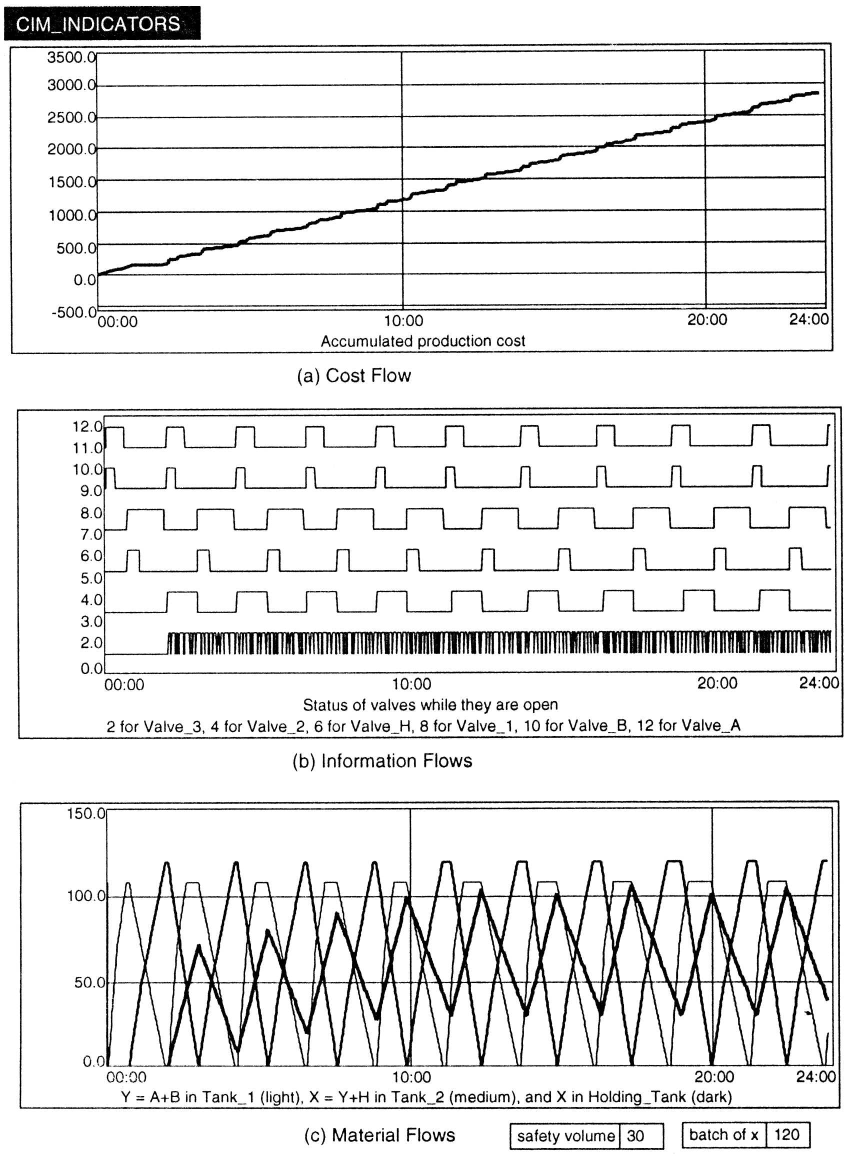


Fig. 21. Indicators of Mold Filling System.

depicts the system's configuration. The PMS at lower right is specified by the VMS in 1/8 Scale, and its components are shown in Fig. 3. Materials A, B, and H are pumped up from the raw material tanks. Each on/off valve is incorporated with a turbine flow meter for *volume_control*. The volume in *Holding Tank* is calculated from *Level Indicator*. Molds are shipped by *Conveyor*. As Fig. 20 indicates, the on/off valves, mixers, motor, and reader on/off port are connected to the Allen–Bradley PLC-5/11 output module; the flow meters, level indicator, sensor, and barcode read-in port are connected to the PLC's input module. A Control Coprocessor 1771 of the PLC is connected via Ethernet to workstation Sun Sparc 10 where a decision system and VMS are designed as workspaces of real-time expert system G2 [18].

The one-to-one mapping between the VMS and PMS is developed through a TCP socket pair based on the client and server model [25]. The server process is in OS-9 (Coprocessor), and the client process is in UNIX (Sun Sparc 10). Through the socket pair, a communication channel is first established. Information packages are then exchanged. From one direction, the VDs pass their output attributes to the server process which subsequently updates the PLC data table. The valves, mixers, motor, and barcode reader in PMS reflect the corresponding VDs' behavior. From the other direction, input module not only scans factors such as each valve's flow volume, the volume in *Holding Tank*, the input signal from *Sensor*, but also refreshes the PLC data table. These values are then fed back to workstation for updating VDs' inputs.

According to the supervisor design steps, from Figs. 4 and 6–9, two rule-based supervisors (KM_1 and KM_2) are designed. The control rules [13] follow DM_1 in Fig. 9 and DM_2 in Fig. 8. In those rules, the virtual devices listed in Table 2 are used. Twelve rules of KM_1 issue commands to control the valves and the mixers for preparing X; six rules of KM_2 displayed in Fig. 10 issue commands to *Conveyer, Reader* and *Valve 3* for mold filling process.

Fig. 21 summarizes the operational results of supervised *Mold Filling System* as illustrated by indicators including cost, control information, and material. Fig. 21a depicts dynamic operational cost vs. time; Fig. 21b displays the information flows which indicate the dynamics of six valves vs. time; Fig. 21c shows the material Y in *Tank 1*, X in *Tank 2*, and X in *Holding Tank* vs. time.

According to the monitor design steps, the worlds illustrated in Fig. 11 are specified with system dynamics shown in Fig. 12. A world-based monitor for DM_2 is defined in Fig. 13. Those symptom detecting rules are shown in Fig. 14. Once a symptom is detected by those rules, the monitor invokes a troubleshooter.

The troubleshooter locates the fault source and recovers the system when it is invoked. According to the troubleshooter design steps, fault diagnosis and recovery plans are designed. If the source causing a symptom is unique, it is immediately determined from Table 3 and 4. For multiple probable sources, the probability of each possible source is computed through Bayes' technique as presented in Section 5.3. Auto-recovery plans follow the summary in Fig. 18, and some of them are displayed in Fig. 16. Aid-recovery plans based on Table 6 are designed and displayed in Fig. 19.

With the aid of the monitor and troubleshooter, the reliability of mold filling process is significantly enhanced when compared with only applying supervisor. Fault propagation is prevented by symptom detecting rules. Auto-recovery plans prolong the mean time to failure (MTTF) by approximately five folds in experimentation, because the recoverable faults are automatically solved. Also, fault diagnosis and aid-recovery plans shorten the mean time to repair (MTTR).

7. Conclusions

In this study, we have proposed a new methodology to build a reliable AMS. The methodology can be applied to functionally decomposable, discrete-event systems such as *Mold Filling System* for supervisory control, monitoring, fault diagnosis, and error recovery.

At the supervisor design stage, rules representing control logic are systematically obtained from transformation. The attributes of VDs composing a VMS are either computed from Petri nets or analyzed from constraints. Moreover, the computed attributes can be used to represent worlds. At the monitor design stage, the system dynamics of a Petri net and first principles in worlds are used to detect symptoms. At the troubleshooter design stage, faults are located by applying deep and shallow knowledge. In addition, recovery plans enhance the reliability in MTTF and MTTR. At the implementation stage, a PMS is installed according to the VMS specifications. Connecting VMS with PMS via a communication network allows a decision system to supervise, monitor, and troubleshoot a remote PMS.

The implemented *Mold Filling System* at Hsin-Chu can be operated and troubleshot under digital images in Taipei via Ethernet (10 Mbps) and FDDI (100 Mbps) [26]. The distance is about 80 km. Results in this study demonstrate the proposed methodology's effectiveness in the design and implementation of a reliable AMS for batch process and discrete manufacturing.

References

[1] T.O. Boucher, M.A. Jafari, Design of a factory floor sequence controller from a high level system specification, J. Manuf. Syst. 11 (1992) 401–417.

[2] K.-P. Brand, J. Kopainsky, Principles and engineering process control with Petri nets, IEEE Trans. Automatic Control 33 (1988) 138–149.

[3] E. Pierard, Reference architecture for car assembly monitoring, Comput. Industry 27 (1995) 203–213.

[4] P. Higgins, J. Browne, The monitor in production activity control system, Production Planning and Control 1 (1989) 17–26.

[5] L. Lia, R. Govind, Development of a process diagnosis scheme using AI techniques, AICHE Symp. Series Processing, Sensing and Diagnostics 85 (1989) 30–35.

[6] D.N. Batanov, Z. Cheng, An object-oriented expert system for fault diagnosis in the ethylene distillation process, Comput. Industry 27 (1995) 237–249.

[7] S.M. Alexander, W.Y. Lee, J.H. Graham, Design-based diagnosis, Int. J. Production Res. 31 (1993) 2087–2096.

[8] M.C. Zhou, F. Dicesare, Adaptive design of Petri net controllers for error recovery in automated manufacturing systems, IEEE Trans. Syst., Man, Cybernetics 19 (1989) 963–973.

[9] A. Visser, An exception-handling framework, Int. J. Comput. Integr. Manuf. 8 (1995) 197–203.

[10] D. Gracanin, P. Srinivasan, K. Valavanis, Time and error recovery with parameterized Petri nets, Proceedings of the 1993 Int. Sympos. on Intelligent Control, Chicago, IL, USA, Aug. 1993, pp. 291–297.

[11] Y.B. Moon, C.L. Moodie, A framework for failure recovery in a manufacturing cell, Int. J. Adv. Manuf. Technol. 4 (1989) 144–156.

[12] G.R. Liang, T.Y. Tseng, Rule-base troubleshooter design for the maintenance of manufacturing devices, Int. IEEE/IAS Conf. on Industrial Automation and Control: Emerging Technology, R.O.C., May 1995, pp. 293–300.

[13] G.R. Liang, H.M. Hong, Hierarchy transformation method for repetitive manufacturing system: specification, design, verification, and implementation, Comput. Integr. Manuf. Syst. 7 (1994) 191–205.

[14] R. Joannis, M. Krieger, Object-oriented approach to the specification of manufacturing systems, Comput. Integr. Manuf. Syst. 5 (1992) 133–145.

[15] J.L. Peterson, Petri Net Theory and the Modelling of Systems, Prentice-Hall, NJ, 1981.

[16] T. Murata, Petri nets: properties, analysis, and applications, Proc. IEEE 77 (1989) 541–580.

[17] R. David, H. Alla, Petri Nets and GRAFCET, Prentice-Hall, UK, 1992.

[18] Gensym, G2 Reference Manual, Version 3.0, USA, 1992.

[19] R. Reiter, A theory of diagnosis from first principles, Artif. Intelligence 32 (1987) 57–95.

[20] J. de Kleer, J.S. Brown, Qualitative physics based on confluences, Artif. Intelligence 24 (1984) 7–83.

[21] V.S. Srinivasan, M.A. Jafari, Fault detection/monitoring using time Petri nets, IEEE Trans. Syst., Man, Cybernetics 23 (1993) 1155–1162.

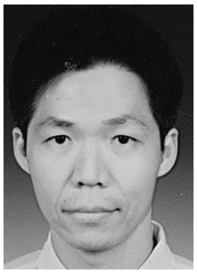
[22] J. Prock, A new technique for fault detection using Petri nets, Automatica 27 (1991) 239–245.

[23] J. Mendigutxia, P. Zubizarreta, J.M. Goenaga, L. Berasategui, L. Manero, Fault tolerance in automated manufacturing systems, Expert Syst. Appl. 8 (1995) 275–285.

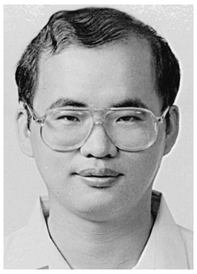
[24] J. Pearl, Probabilistic Reasoning in Intelligent Systems: Network of Plausible Inference, Morgan Kaufmann Publishers, San Mateo, CA, 1988.

[25] W.R. Stevens, UNIX Network Programming, Prentice-Hall, 1990.

[26] G.R. Liang, Advent of network-integrated manufacturing age in Taiwan, The 5th Int. Conf. on Automation Technology, Taiwan, R.O.C., July 1996, pp. 29–36.



Woei-Horng Jeng is currently Associate Professor in Department of Industrial Engineering and Management at Ming Hsin Institute of Technology, Taiwan, R.O.C. He received the Doctor of Philosophy degree in Industrial Engineering from National Chiao Tung University, R.O.C, in May 1997, his Master of Science degree in Industrial Engineering from National Tsing Hua University, R.O.C, in 1987, and his bachelor degree from National Taiwan University in 1980. He was also a senior engineer in the design of manufacturing system and material handling system at MIRL, ITRI, R.O.C. His research interests are automated manufacturing system design, computer-integrated manufacturing, intelligent manufacturing systems, and shop floor control.



Gau Rong Liang is Associate Professor in Institute of Industrial Engineering at National Chiao Tung University, Taiwan, R.O.C. Also he had worked as researcher in Bell Labs., Columbus, Ohio, USA. in 1984 and in INRIA-Lorraine, Metz, France in 1990. His current research interests are computer-integrated manufacturing, shop floor control system design, and intelligent manufacturing execution systems. He received the Doctor of Philosophy degree in School of Industrial Engineering at Purdue University, IN, USA in 1987, his Master of Science degree in Institute of Electrical Engineering from National Taiwan University in 1981, and his Bachelor of Science degree in Department of Control Engineering from National Chiao Tung University in 1979. He won Outstanding Young Industrial Engineer Award from CIIE and national Outstanding Information Technology Application Award in 1995.