Fault Recovery in Control Systems:

A Discrete Event System Approach

Mohammad Moosaei

A Thesis

in the Department of

Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements

For the Degree of Master of Applied Science at

Concordia University

Montreal, Quebec, Canada

August 2003

Your file Votre référence
ISBN: 0-612-83873-0
Our file Notre référence
ISBN: 0-612-83873-0

Abstract

Fault Recovery in Control Systems: A Discrete Event System Approach

Mohammad Moosaei

Fault recovery is a challenging task that is crucial in achieving stringent reliability and safety goals. In this thesis, the problem of fault recovery is studied in discrete-event systems (DES), assuming permanent failures.

A diagnosis system is assumed to be available to detect and isolate faults with a bounded delay. Thus, the combination of the plant and diagnosis system can be thought of having three modes: normal, transient, and recovery. Initially the plant is in the normal mode. Once a failure occurs, the system enters the transient mode. After the failure is diagnosed by the diagnosis system, the system enters the recovery mode.

This framework does not depend on the diagnosis technique used, as long as the diagnosis delay is bounded. As a result, the diagnosis and control problems are almost decoupled.

In general, for each mode there is a set of specifications that have to be met. We propose a modular switching supervisory scheme. The proposed framework contains one normal-transient supervisor and multiple recovery supervisors each corresponding to a particular failure mode. Once a fault is detected and isolated by the diagnoser, the normal-transient supervisor is removed from the feedback loop and one of the recovery supervisors will take sole control of the system. The issue of non-blocking is studied and it is shown that essentially if the system under supervision is non-blocking in the normal mode, then it will remain non-blocking during the recovery procedure. Supervisor admissibility is also studied.

This approach is developed for untimed DES and then extended to timed DES. In the process, previous results on supervisor design for untimed DES with partial observation are extended to timed DES. Various examples from manufacturing and process control are provided to illustrate the approach.

Acknowledgements

I would like to gratefully thank my supervisor, Professor Shahin Hashtrudi Zad for his support, guidance, and encouragement throughout this research. I thank the faculty and the staff of the electrical and computer engineering department for providing me an environment to learn and to research.

I would like to thank my wife, Sanaz. My achievements, success, and happiness would not be possible without her. I also would like to thank both of our families for their support.

I would like to dedicate this work to my sister, Zarafshan and my sister-in-law, Esmat who they passed away during this research.

Table of Contents

List of Figures

List of Tables

Chapter 1

1.1 Introduction

Providing fault recovery is one of the true challenges in designing highly reliable systems. It is important to maintain the performance requirements of systems even during systems faults. This thesis studies the problem of fault recovery using discrete-event models.

Information systems, manufacturing systems, air-traffic management, and communication protocols, are all examples of discrete-event systems (DES). It is desirable to restrict the behavior of a given DES to comply with a given set of specifications. Supervisors (controllers) are designed to enforce suitable restriction on the behavior of the DES. For example, in the area of manufacturing, the supervisory control involves the control and coordination of a number of interacting machines or groups of machines and processes to ensure that the production proceeds according to some recipe

and that the system does not become trapped in a deadlock. This supervisory control may be at the *work-cell-shop-floor level* or may be at the *work-cell/production-line level.* In the area of communication protocols, supervisory control can be applied to regulate the communication network behavior.

The system or process to be controlled is typically called the *plant.* We assume the plant can be modeled as a *Finite State Automaton* (FSA). Each FSA specifies a desired behavior (language). Fig. 1.1 shows a supervisor **S** in a control feedback position with the plant **G**. The supervisor monitors the events unfolding in the plant and sends appropriate commands to the plant to ensure the generated event sequence satisfies the design specifications.

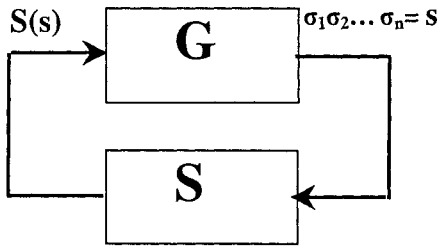


Figure 1.1: Supervisory control loop

Since safe and reliable operation has always been of paramount importance in engineering systems, fault detection, isolation, and recovery mechanisms are essential components.

Any non-permitted changes in the behavior of a system leading to degradation in the overall system performance for a bounded or unbounded period of time under specified operating conditions is called *fault.* In this thesis, the terms *fault* and *failure* have the

same meaning and they are used interchangeably. A fault may be the result of a malfunction of an actuator or a sensor in the system, or a malfunction in a system component. An example of failure in a component is a valve becoming stuck in its closed position, not permitting the fluid to flow through. Other examples of failure include a tank leaking or the failure of the power supply of a PLC.

Faults can be divided into two categories: permanent and nonpermanent. A fault is said to be permanent if, after it occurs, the system remains in a faulty condition indefinitely. Otherwise it is nonpermanent, and after occurring at a particular time, it stays in the system for some period of time, and then disappears. An example of a permanent fault is a broken shaft in a motor, and an example of a nonpermanent fault is a loose wire in an electrical system.

For fault recovery in DES, a few frameworks have been proposed and some are in use ([1], [2], [3], and [8]).

In [1], the system DES model is analyzed in order to classify faults and failures quantitatively and to find a "fault tolerant" event sequences which can take the system from the initial state to a marked state despite the occurrence of faults. The proposed approach has two steps: conducting failure analysis on the discrete-event model of the system followed by constructing the fault-tolerant supervisor. The failures are modeled as abnormal events leading to faulty states. The synthesis of the fault-tolerant supervisor is based on the results of the previous failure analysis. A work procedure is called fault-tolerable if a) the technique can find an event sequence that can reach a marked state, or

b) the technique can eliminate the path to the abnormal events. This approach does not provide any solution if the specifications need to be changed after the occurrence of faults and, also, does not address fault recovery directly. Moreover, this method does not provide any solution if the proposed scheme cannot find an event sequence to marked states.

Another approach to tackle the problem of uncertainty or subsystem failure is proposed in [3]. The authors assume that the plant is a complex system composed of several subsystems. If a subsystem fails, then a learning mechanism deletes the events that pertain exclusively to the failed subsystem from the desired language. After that, it applies a specific automaton repair algorithm in order to patch the desired language. Finally, this approach computes a new supervisor aimed at restricting the behavior of the remaining system to a new desired language. It seems that this approach is suitable for cases in which the faulty subsystem is to be shutdown. However, if the recovery goal is to replace the faulty subsystem by a backup subsystem or if the specifications are changed, then this approach may not be applicable.

In [2], qualitative representations are developed in model-based reasoning and coupled with the transition systems models underlying concurrent reactive languages. This approach, which was used in NASA's Deep Space 1 spacecraft, infers the current state of the system from its knowledge of the previous state and from observations at the current state. Then it considers actions that achieve the configuration goal within the next state. In other words, they introduce a model-based configuration manager that uses a

specification of the transition system to compute the desired sequence of control commands. This approach allows the system to move among desirable states satisfying the specifications. However, the disadvantage of this methodology is that it only estimates the next state of the system in each action, and it does not consider multiple transitions. This may limit the efficiency of control and recovery procedures.

In [8], the authors present an integrated approach to control and diagnosis. They study the active diagnosis problem in the framework of DESs. Since in this case, control and diagnosis problems are integrated, the controller design has, in the worst case, exponential complexity in the number of plant states.

In this thesis, we study the problem of the synthesis of recovery procedures in discrete-event systems. An overview of the thesis is provided in the following section.

1.2 Thesis Outline

In Chapter 2, we briefly review supervisory control, in particular, the Ramadge-Wonham Framework. We use this theory to formulate and solve supervisory control problems. This chapter covers both untimed and timed discrete-event systems. Moreover, since we are dealing with a supervisory control problem under partial observation, the concept of observability, normality, and controllability are discussed in both timed and untimed cases. Finally, some techniques proposed in the literature for fault diagnosis are briefly reviewed.

It is assumed that the plant can be modeled as a finite state automaton describing the system behavior in both normal and faulty conditions. We assume there are p permanent failure modes, F_1, ..., F_P. For our work, the availability of models that accurately describe the dynamics of system behavior in normal and faulty conditions is very crucial in the synthesis of fault recovery procedures.

In Chapter 3, we present our framework for designing fault recovery procedures in untimed DES. In our framework, a new block modeling the diagnosis system (diagnoser) is added to the plant. It is assumed that the diagnoser detects and isolates events with a bounded delay and then reports it to the supervisor. Thus, the system to be controlled which consist of the plant and the diagnosis system will have three modes: *normal*, *transient*, and *recovery*. In the normal mode, the plant is functioning properly. Once a failure occurs, the system enters its transient mode. At this point, the failure has not been diagnosed yet. Once the failure is diagnosed the system enters the recovery mode.

The diagnosis system is modeled as finite state automaton that generates a detection event with some delay after a failure. One of the advantages of this approach is that the diagnosis and control problems are almost decoupled. Note that in our work it is not important what diagnosis technique is used by the diagnosis system as long as the diagnosis delay is bounded.

In general, for each mode (normal, transient, and recovery), the system has a set of specifications. We call the specifications of the normal, transient, and recovery modes *normal specifications*, *transient specifications*, and *recovery specifications*, respectively.

Following a modular switching approach, we construct a specific supervisor (controller) for each mode. Initially, all three supervisors are in the feedback loop but after the detection and isolation of a fault, a recovery supervisor will take sole control of the plant.

We would like the plant under supervision to be non-blocking during its normal operation and in case of failure, be non-blocking during the recovery procedures. Loosely speaking, we show that if the system under supervision is non-blocking in normal mode, it will also be non-blocking during the recovery procedure.

In the modular switching supervisory control approach used here, the resulting supervisor may not be admissible even if all modules are admissible. We examine the problem of supervisor admissibility and provide remedies in cases where the designed supervisor is not admissible.

Examples are used to demonstrate the ideas and the procedures developed.

In Chapter 4, we extend our approach to fault recovery in timed DES. We add a clock tick event to the plant event set and assume the plant can be modeled as a timed DES.

Since faults are assumed unobservable, the problem of finding suitable controllers for fault recovery is an instance of the problem of supervisory control under partial observation. In order to synthesize supervisors for these problems, we extend an existing procedure in the literature for supervisor design using normal languages for untimed DES to timed DES.

In Chapter 5, we apply our framework to a manufacturing system. Finally, in the last chapter, we summarize the results of the thesis and discuss topics for future research.

Chapter 2

Background

2.1 RW Supervisory Control Theory

In [27] Ramadge and Wonham (RW) introduced a framework to model and construct supervisory controllers for DES. In this framework, given a plant and a set of specifications (both in the form finite state automata), the objective is to design a supervisor (controller) for a plant so that the plant under control (the closed-loop system) satisfies the specifications. Some advantages of using this theory are as follows:

i. It is structured. That is, to construct the controller one needs to obtain the plant and specification finite automata and then input them to the RW algorithm. In return, the algorithm generates a unique control structure which is non-blocking and maximally permissive.

ii. It treats the open-loop plant and the controller separately where as other approaches only deal with models of the closed-loop system.

iii. The method is general enough to cover different types of control specifications.

The main challenge in this framework is the computational complexity for large-scale systems. However, this problem may be mitigated by adopting such techniques as decomposition and distributed modeling and control. Furthermore, the significance of computational complexity problem is reduced due to the fact that all the necessary calculation can be performed off-line. In this section a brief review of this theory is provided. For details, the reader is referred to [26] and [27].

Finite State Automaton

It is assumed that the plant can be modeled as a finite state automaton (FSA)

$$G = (Q, \Sigma, \delta, q_\circ, Q_m)$$

where Q is the set of states, $\Sigma = \Sigma_c \cup \Sigma_u$ is the event set, where Σ_c and Σ_u are the sets of *controllable* and *uncontrollable events*, respectively, with $\Sigma_c \cap \Sigma_u = \Phi$. δ is the transition partial function $\delta: \Sigma \times Q \rightarrow Q$, $q_\circ$ is the initial state, and $Q_m \subseteq Q$ is the set of marked states. Let $L(G)=\{s \in \Sigma^* \mid \delta(s, q_\circ)$ is defined$\}$ be the uncontrolled language generated by G over alphabet Σ or the "closed behavior" of G. L(G) is the set of all possible sequence of events that take G from the initial state to some reachable state.

$L_m(G)$ is the marked behavior generated by the uncontrolled process G; i.e., the set of all possible sequence of events which take the initial state to some marked state in G. By

uncontrolled language, we refer to the behavior of G as an event generator when it is not subject to any control. A controllable event is one that can be enabled or disabled at any time by some means of control. Any event that is not controllable is considered uncontrollable.

Synchronous and Parallel Products

In the study of DES, we often have to build complex models using the individual models of system components. Two operations that are particularly useful in modeling the joint operations of DES are "parallel product" and "synchronous product".

Consider two DES G1 and G2 with alphabets Σ_1 and Σ_2. In parallel product, the two DES synchronize the occurrence of their common events: $\sigma \in \Sigma_1 \cap \Sigma_2$ occurs if it is defined and enabled in both G1 and G2. Events not in $\Sigma_1 \cap \Sigma_2$ are disabled.

In synchronous product, the DES synchronize on the common events (similar to parallel product). However, each may execute its own private events without any synchronization, that is events in $\Sigma_1 - \Sigma_2$ and $\Sigma_2 - \Sigma_1$. The reader may refer to [26] for details.

TTCT is a software program which is used for analysis, synthesis, and verification of supervisory controls. [22] provides a brief review of TTCT functionality and procedures. Some of the TTCT procedures that are used in this thesis are reviewed in this chapter.

The TTCT **sync** procedure forms the synchronous product of G1 (with event set) and G2 (with event set Σ_2) to create G3. The event set G3 will be $\Sigma_1 \cup \Sigma_2$.

G3 = **sync**(G1, G2).

The **meet** procedure forms the (reachable) parallel product of G1 and G2 to create G3 with $\Sigma_3 = \Sigma_1 \cap \Sigma_2$.

$$G3 = \textbf{meet}(G1, G2).$$

Non-blocking

Non-blocking is one of the important properties of DES. A DES is said to be *non-blocking* if $\overline{L}_m(G) = L(G)$, where $\overline{L}_m(G)$ is the prefix closure of $L_m(G)$. In other words, G is non-blocking if for any string $t \in L(G)$, there is at least one string s such that $ts \in L_m(G)$. This means that from every reachable state in G, there is a path to a marked state.

Specifications

Usually, we represent the desired behavior (or specifications) of the system in the form finite state automata. Let E be such DES. Then $L(G) \cap L(E)$ represents all of the system's legal or desirable (closed) behavior. $L(G) \cap L(E)$ is called the legal behavior.

Supervisor

A supervisor is an agent that enables or disables controllable events such that the discrete event system G generates a language that satisfies the specifications. In other words, it prevents G from generating undesirable event sequences. In the cases of interest to us, the supervisor S may be modeled as a finite state automaton

$$S = (X, \Sigma, \eta, x_o, X_m)$$

where X is the state set, x_o *is* the initial state, $X_m \subseteq X$ is the set of marked states, and $\eta : \Sigma \times X \rightarrow X$ is the transition function. A supervisor should never attempt to disable uncontrollable events (because this is not possible by assumption).

It is assumed that S is a standard supervisor; $X = X_m$, that is, all the supervisor states are marked. Therefore, marking is done by the plant. Let L(S/G) denote the language generated by G under supervision of S.

The Coupled Plant and Supervisor:

According to RW theory, the supervisor and the process are coupled to form a closed – loop system. This feedback mechanism between S and G functions as follows. Suppose the process is in state q_i and the supervisor is in state x_j at a given time. A subset of events σ can occur in the uncontrolled process in state q_i. This subset of events is fed into the supervisor. In state x_i (based on the specification language) only a subset of these events may be permitted. Then the supervisor issues a control pattern such that the controllable events are enabled if they are permitted at x_j and disabled if they are not. The idea is illustrated by Fig. 1.1, and the closed loop system under control is denoted by S/G.

The following languages generated by S/G are of interest.

i. The closed behavior of S/G which is defined by $L(S/G) = L(G) \cap L(S)$.

   L(S/G) consists of the sequence of events of uncontrolled process language that survive under supervision.

ii. The marked behavior of S/G which is defined by $L_m(S/G) = L_m(G) \cap L_m(S)$.

L_m(S/G) consists of the sequence of events that are marked by both G and S and survive under supervision.

We have to ensure that the supervisor does not disable an uncontrollable event. This can be described using the property of controllability.

A language K is said to be controllable with respect to L(G) and Σ_u if $\sigma \in \Sigma_u$, $s \in \overline{K}$, $s\sigma \in L(G)$ implies $s\sigma \in \overline{K}$.

There we always require that L(S/G) be controllable with respect to L(G) and Σ_u.

The RW algorithm accepts the DES G and specification models, and generates a minimally restrictive supervisor S. i.e., the language L(S/G) is the supremal controllable sublanguage of L(H) (legal behavior) which can be generated through the control mechanism.

S is called a non-blocking supervisor if $\overline{L}_m(S/G) = L(S/G)$.

Computation of Supervisor:

Given a plant G and a specification automaton E, to obtain the proper controller (supervisor), using the RW supervisory control theory, we compute the supremal controllable sublanguage of the legal marked language $L_m(H)=L_m(G) \cap L_m(E)$. This computation can be done by the TTCT procedure **supcon,** which returns a trim automaton representing the supervisor (S).

S = supcon(G , E)

Verification of Supervisors:

We briefly review the procedure for verifying whether a given supervisor is proper and that it satisfies the plant specifications [26]. We have to check the following four conditions.

I. The supervisor is admissible: We have to see whether the closed behavior of the supervisor L(S) and there L(S/G) = L(S) ∩ L(G) is controllable with respect to the closed behavior of the plant (L(G)) and the uncontrollable events (Σ_u). This means that the supervisor will not attempt to disable an uncontrollable event.

Condat is a TTCT procedure used to verify the admissibility of a supervisor.

SDAT = condat(G , S)

This procedure generates a list of supervisor states in which disabling occurs, together with the events that must be disabled. If L(S) is controllable, then the list will include only controllable events.

II. The plant under supervision is non-blocking:

The TTCT procedure used for investigating the non-blocking property is called **nonconflict**.

nonconflict(G , S) = true ?

If the answer is "true", then S is a non-blocking supervisor, meaning that the plant under supervision (S/G) is non-blocking.

III. The supervisor is trim, i.e., all supervisor states must be reachable and coreachable. To investigate this property, the following TTCT procedures are applied:

TRIMS = trim(S) , **isomorph(S , TRIMS)**

Trim is used to obtain the reachable and coreachable part of the DES, while **isomorph** is used to check if two DES's are identical up to renumbering of states. In this thesis, all supervisor states are reachable and marked. Therefore, the supervisors are trim.

IV. The plant under supervision satisfies the specifications.

$$L(S/G) \subseteq L(Spec) \quad , \quad L_m(S/G) \subseteq L_m(Spec)$$

For supervisors computed by the RW theory the satisfaction of the specification has already been taken into account in the supervisory design procedure. Therefore, it is not necessary to check this property. In other words, the design satisfies the specifications by construction.

To verify the above equations by TTCT, the following equality can be checked:

trim(meet(S/G , complement(Spec,--))) = EMPTY

Here, **complement** is a procedure that produces an automaton whose marked language is the complement of the marked language of **Spec** (The complement of a language $L \subseteq \Sigma^*$ is $\Sigma^* - L$).

Given a set of specifications, we can design a supervisor for each specification separately and form a supervisor to satisfy all specifications by conjunction of smaller supervisors. This is referred to as modular supervision design. The concept of nonconflicting languages is often encountered in the study of modular supervisors.

Nonconflicting: Let L1, L2 be arbitrary sublanguages of Σ^*. Then L1, L2 are nonconflicting if

$$\overline{L1 \cap L2} = \overline{L1} \cap \overline{L2}$$

In TTCT, procedure **nonconflict** is a boolean function that verifies the nonconflicting property for two languages. If **nonconflict**(G1,G2) = true , it means that every reachable state of the product structure **meet**(G_1,G_2) is coreachable or $\overline{Lm(G1) \cap Lm(G2)} = L(G1) \cap L(G_2)$. Therefore, if G1 and G2 are non-blocking, then the procedure nonconflict can be used to see if L_m(G1) and L_m(G2) are nonconflicting.

We may extend the definition of nonconflicting languages to multiple languages in the following way.

Let L1, L2, ..., Lp be arbitrary languages. Then L1, L2, ..., Lp are nonconflicting if

$$\overline{L1 \cap L2 \cap ... \cap Lp} = \overline{L1} \cap \overline{L2} \cap ... \cap \overline{Lp}$$

Partial Observation

The word "observation" is used to describe the information flow from plant to controller. Under a full observation policy, the controller should be able to observe all events

generated by the plant and make the appropriate control decision after each observation. A partial observation policy is referred to the situation when controller observes only part of event set, and makes the appropriate control decision. After each observation, the controller feeds back the prescribed control decision to the plant such that the plant behavior is kept within the specification. It should be noted that such a policy must be considered at design step as well if it is to be used in the implementation.

Lin and Wonham (LW) introduced the concept of observable and normal languages in the study of control under partial observation [14]. They conclude that observability is the essential and therefore, the minimum requirement for the existence of a controller with partial observation. However, it is proven that the set of observable languages is not necessarily closed under union. Moreover, they consider the normality as a sufficient condition for the same purpose. Normal languages are closed under union and so they can be used to derive a systematic algorithm for the controller design. Here, we briefly review the main concepts in the theory of supervisory control with partial observations.

Natural Projection

Let Σ_o be the set of observable events. The natural projection $P: \Sigma^* \rightarrow \Sigma^*_o$ is defined according to

$$\begin{cases} P(\varepsilon) = \varepsilon \\ P(\sigma) = \sigma \quad \text{if} \quad \sigma \in \Sigma_o \\ P(\sigma) = \varepsilon \quad \text{if} \quad \sigma \in \Sigma - \Sigma_o \\ P(s\sigma) = P(s)\,P(\sigma) \quad \text{for } s \in \Sigma^*,\ \sigma \in \Sigma \end{cases}$$

where ε is a null string. Thus, the effect of P on a string s is just to erase from s the events that do not belong to Σ_o, leaving the order of Σ_o-events in s unchanged.

Let $A \subseteq \Sigma^*$, $B \subseteq \Sigma^*_o$. It can be shown that

$$\overline{PA} = P\overline{A} \quad ; \quad \overline{P^{-1}B} = P^{-1}\overline{B}$$

The usual inverse image function of P is

$$P^{-1}: P_{wr}(\Sigma^*_o) \rightarrow P_{wr}(\Sigma^*)$$

where P_{wr} is the power set.

Note that in supervisory control under partial observation, control decisions are based on the projection of event sequences (P(s)). We expect that if two strings s and s' have the same projection (P(s) = P(s')), then corresponding supervisory decisions be the same. Such a supervisor is called "feasible".

Observability and Normality

K is said to be **(L, P)-normal** or simple **normal** with respect to L, if $K = L \cap P^{-1}P(K)$. In other words K is called normal if and only if it can be recovered from its projection along with knowledge of the structure of L. An equivalent mathematical description of normality is as follow:

$$s \in K, s' \in L, P(s) = P(s') \Rightarrow s' \in K$$

A language K is ***(G, P)–observable*** if for all s, s' with P s = P s', we have the following:

$$\begin{cases} (\forall\ \sigma)\ s\sigma \in \overline{K}\ \&\ s' \in \overline{K}\ \&\ s'\sigma \in L(G) \Rightarrow s'\sigma \in \overline{K} \\ s \in K \cap L_m(G)\ \&\ s' \in \overline{K} \cap L_m(G) \Rightarrow s' \in K \end{cases}$$

Suppose $K \subseteq L(G)$ and K is closed, or $K \subseteq L_m(G)$ and $K= \overline{K} \cap L_m(G)$, then $\overline{K}$ is (L(G), P)-normal implies K is (G,P)-observable [14].

Let $K \subseteq L_m(G)$ and nonempty. The necessary and sufficient conditions for the existence of a ***feasible supervisor*** control S for G such that $L_m(V/G) = K$ are:

i) K is Lm(G)- closed (i.e., $K= \overline{K} \cap L_m(G)$)

ii) K is controllable with respect to G

iii) K is observable with respect to (G, P)

Because the set of languages defined by the above conditions is not necessarily closed under union, there is no an optimal (minimally restrictive) supervisory control. This makes supervisor design based on observable languages difficult.

In [26], it is demonstrated that a subset of normal languages is closed under union.

Let $E \subseteq L_m(G)$ be a specification language. The goal of the supervisory control and observation problem (SCOP) is to find a non-blocking, feasible, and admissible supervisor R such that

$$\Phi \neq L_m(R/G) \subseteq E$$

Let

$$S(E) = C(E) \cap N(E; L_m(G)) \cap \overline{N}(E; L(G))$$

where

$$C(E) = \{ K \subseteq E \mid K \text{ is controllable with respect to } G\}$$

$$N(E; L_m(G)) = \{ K \subseteq E \mid K \text{ is } (L_m(G), P)\text{-normal}\}$$

$$\overline{N}(E; L(G)) = \{ K \subseteq E \mid \overline{K} \text{ is } (L(G), P)\text{-normal}\}$$

Then $S(E)$ is nonempty and closed under arbitrary unions. The following theorem has been proved in [26] in order to find a solution for SCOP.

Theorem:

Let $K \neq \Phi$ and $K \in S(E)$.

Then define

$$R: L(G) \rightarrow \Gamma$$

with

$$R(s) = \begin{cases} \Sigma_u \cup \{ \sigma \in \Sigma_c \mid P(s\sigma) \in P\overline{K} \} & Ps \in P\overline{K} \\ \Sigma_u & Ps \in PL(G) - P\overline{K} \end{cases}$$

Solves SCOP with

$$L_m(R/G) := L(R/G) \cap K$$

Furthermore, $L_m(R/G)=K$.

LW Procedure for Computing Supervisor Under Partial Observation

Assume G is a non-blocking plant, and let E be the specification in the SCOP. Then the following procedure can be used to compute a supervisor [26].

0. **Given G, E, and the list NULL of P-unobservable events**
1. **N = supnorm(E , G , NULL)**
2. **NO = project(N , NULL)**
3. **GO = project(G , NULL)**
4. **KO = supcon(GO , NO)**
5. **KODAT = condat(GO , KO)**
6. **PINVKO = selfloop(KO , NULL)**
7. **nonconflict(G , PINVKO)** = true **?**
8. **K = meet (G , PINVKO)**
9. **K** nonempty **?**

Here, supnorm computes a DES N with $L_m(N) = \sup N\ (L_m(E) \cap L_m(G);\ L_m(G))$. If the answers to questions 7 and 9 are "yes", then **PINVKO** is a solution to the supervisory control and observation problem (SCOP). The plant under supervision denoted by **K** is the corresponding controlled behavior.

2.2 Timed Discrete Event Systems

In Timed Discrete Event Systems (TDES), the event set includes a tick event. The occurrence of non-tick events is restricted by lower and upper time bounds.

The tick represents the progress of time on a global clock. This event has no lower or upper time bound. A set of forcible events may be applied in order to preempt the tick event. These events will be discussed later.

We use the theory of TDES described in [11]. This theory starts with a definition of the Activity Transition Graph (ATG) model of a TDES. An ATG is modeled by a five-tuple

$$Gact = (A , \Sigma_{act} , \delta_{act} , a_{\circ} , A_m)$$

Here, A is the activity set, Σ_{act} is a finite set of events, δact: $A \times \Sigma_{act} \rightarrow A$ is the activity transition partial function, $a_{\circ}$ is the initial activity, and A_m is the subset of marked activities. The events and the activities differ in that events are instantaneous, while activities involve a time duration. Consider *a* and $\sigma \in \Sigma_{act}$ to be an activity and an event of the above ATG. An event σ is enabled at *a* if $\delta_{act}(a., \sigma)$ is defined.

In this framework, for each non-tick event σ there is a time interval $[l_\sigma,u_\sigma]$, where $l_\sigma \in N$ and $u_\sigma \in N \cup \{\infty\}$ denote the lower and upper time bounds, and $l_\sigma \leq u_\sigma$ (N = {0, 1, 2,...}).

The event set Σ_{act} is partitioned to two categories, *prospective* and *remote*. If $0 \leq u_\sigma \in N$ then the event is called prospective, while if $u_\sigma = \infty$, then it is called remote.

$$\Sigma_{act} = \Sigma_{rem} \cup \Sigma_{spe}$$

A prospective event cannot occur before l_σ clock ticks but will occur before the $(u_\sigma+1)$ clock tick (unless it is preempted by another event in Σ_{act}), while a remote event cannot occur before l_σ clock ticks but may occur any time after the l_σ–th tick.

The timed events are defined as triples $(\sigma, l_\sigma, u_\sigma)$ and the set of these events is denoted by Σ_{tim}.

$$\Sigma_{tim} = \{(\sigma, l_\sigma, u_\sigma) | \sigma \in \Sigma_{act}\}$$

Assume j, k $\in$ N with j $\leq$ k. Define [j, k] as a set of integers between j and k. For each event σ the timer interval (T_σ) is defined as follows:

$$T_\sigma = \begin{cases} [0, u_\sigma] & \sigma \text{ prospective} \\ [0, l_\sigma] & \sigma \text{ remote} \end{cases}$$

For each event $\sigma \in \Sigma_{act}$, the default timer value associated with each σ is

$$t_\sigma = \begin{cases} u_\sigma & \sigma \text{ prospective} \\ l_\sigma & \sigma \text{ remote} \end{cases}$$

where l_σ and u_σ respectively denote the lower and upper time bounds of σ.

Suppose $\sigma \in \Sigma_{act}$ becomes enabled in $a \in$ A. After each subsequent occurrence of the tick, t_σ corresponding to event σ will be decremented by one. This will continue until either σ occurs, t_σ reaches zero, or another event occurs and disables event σ. A remote event can happen at any time if its t_σ becomes zero. In the case of a prospective event, because $0 \leq t_\sigma \leq u_\sigma$, the delay of the occurrence of the event cannot be more than u_σ. Therefore, when timer t_σ becomes zero, then the prospective event σ may occur unless another event from the activity event set occurs. We see that the state of the system is described by its current activity and the current values of the timers. Transition among states can be captured by the timed transition graph (TTG).

ATG models can be transferred into Timed Transition Graph (TTG) that represents a TDES. The TDES automaton G_τ is defined as

$$G\tau = (Q_\tau , \Sigma_\tau , \delta_\tau , q_\circ, Q_m),$$

where $Q_\tau = A \times \prod\{T_\sigma \mid \sigma \in \Sigma_{act}\}$ is the state set, $\Sigma_\tau = \Sigma_{act} \cup \{tick\}$ is the event set, δ_τ : $Q_\tau \times \Sigma_\tau \rightarrow Q\tau$ is the state transition partial function, $q_\circ=(a_\circ, \{t_{\sigma\circ} \mid \sigma \in \Sigma_{act}\})$ is the initial state, and $Q_m \subseteq A_m \times \prod\{T_\sigma \mid \sigma \in \Sigma_{act}\}$ is the subset of the marked states. TTG is simply the graph of $G\tau$.

Let $q=(a, \{t_\beta \mid \beta \in \Sigma_{act}\})$, $q'=(a', \{t'_\beta \mid \beta \in \Sigma_{act}\})$, and $\sigma \in \Sigma_{act}$. Then the formal definition of the transition function δ_τ is as follows:

$\delta_\tau(q, \sigma)$ is defined if and only if

1. $\sigma = tick$ and $\forall\ \beta \in \Sigma_{spe}$, $t_\beta > 0$; or
2. $\sigma \in \Sigma_{spe}$, $\delta_{act}(a, \sigma)$ is defined, $0 \le t_\sigma \le u_\sigma - l_\sigma$; or
3. $\sigma \in \Sigma_{rem}$, $\delta_{act}(a, \sigma)$ is defined, $t_\sigma = 0$.

Let $q' \in \delta_\tau(q, \sigma)$.

1. If $\sigma = tick$, then $a' = a$, and

$$\text{If } \beta \in \Sigma_{spe},\ t'_\beta := \begin{cases} u_\beta & \text{if } \delta_{act}(a, \beta) \text{ is not defined} \\ t_\beta - 1 & \text{if } \delta_{act}(a, \beta) \text{ is defined and } t_\beta > 0, \end{cases}$$

(Note that if $t_\beta = 0$, then $\delta_\tau(q, tick)$ in not defined)

$$\text{If } \beta \in \Sigma_{rem},\quad t'_\beta := \begin{cases} l_\beta & \text{if } \delta_{act}(a, \beta) \text{ is not defined} \\ t_\beta - 1 & \text{if } \delta_{act}(a, \beta) \text{ is defined and } t_\beta > 0 \\ 0 & \text{if } \delta_{act}(a, \beta) \text{ is defined and } t_\beta = 0 \end{cases}$$

2. If $\sigma \in \Sigma_{act}$, then $a' = \delta_{act}(a, \sigma)$, and

$$\text{If } \beta \neq \sigma, \beta \in \Sigma_{spe} \quad t'_\beta := \begin{cases} u_\beta & \text{if } \delta_{act}(a', \beta) \text{ is not defined} \\ t_\beta & \text{if } \delta_{act}(a', \beta) \text{ is defined,} \end{cases}$$

$$\text{If } \beta = \sigma, \sigma \in \Sigma_{spe} \quad t'_\beta := u_\sigma$$

$$\text{If } \beta \neq \sigma, \beta \in \Sigma_{rem}, \quad t'_\beta := \begin{cases} l_\beta & \text{if } \delta_{act}(a', \beta) \text{ is not defined} \\ t_\beta & \text{if } \delta_{act}(a', \beta) \text{ is defined,} \end{cases}$$

$$\text{If } \beta = \sigma, \sigma \in \Sigma_{rem} \quad t'_\beta := l_\sigma$$

Example 2.1: Given a simple DES with $\Sigma_{act} = \{\lambda, \beta\}$, and

$$G_{act} = (A, \Sigma_{act}, \delta_{act}, \alpha_\circ, A_m)$$

with $\alpha_\circ = 0$, $A = \{0, 1\}$, $A_m = \{0\}$, and timed events $(\lambda, 1, 1)$, $(\beta, 2, 3)$, both prospective.

The ATG for this DES is

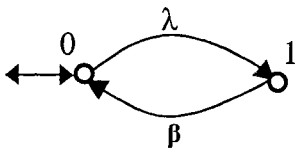


Figure 2.1: Example 2.1. A simple ATG model

Here, the activity transition function is

$$\delta_{act}(0, \lambda) = 1 \text{ and } \delta_{act}(1, \beta) = 0.$$

Fig. 2.2 shows the constructed TTG model of the system (G_τ). The state set for G_τ is

$$Q_\tau = \{0, 1\} \times T_\lambda \times T_\beta = \{0, 1\} \times [0, 1] \times [0, 3].$$

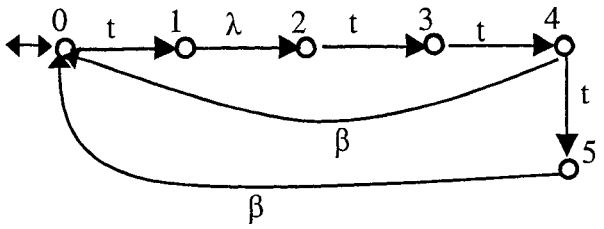


Figure 2.2. Example 2.1. TTG

The event set $\Sigma_\tau = \{\lambda, \beta, t\}$, where t is the tick event, $Q_m = \{(0, 1, 3)\}$, and the list of timer intervals associated to both events λ, β and corresponding to the states ($\{0, 1\}$, $\{t_\lambda, t_\beta\}$) of G_τ are as follows.

State (node of TTG):	0	1	2	3	4	5
Components $[t_\lambda, t_\beta]$:	[1,3]	[0,3]	[1,3]	[1,2]	[1,1]	[1,0]

In the theory of TDES, similar to untimed discrete event systems, the closed behavior of the TDES G_τ is defined to be $L(G_\tau) = \{s \in \Sigma_\tau^* | \ \delta_\tau(q_\circ, s) \text{ is defined}\}$, while the marked behavior of G_τ is $L_m(G_\tau) = \{s \in L(G_\tau) | \ \delta_\tau(q_\circ, s) \in Q_m\}$.

For the purpose of controllability analysis in TDES, the set of events Σ_τ is partitioned into two subsets: *prohibitible* (Σ_{pro}) and *uncontrollable* (Σ_{un}). The former can be disabled and prevented from being in the current list of eligible transitions at a state q of the TDES under supervision.

$$\Sigma_{un} = \Sigma_{act} - \Sigma_{pro} \quad \text{or} \quad \Sigma_{un} = \Sigma_{spe} \cup (\Sigma_{rem} - \Sigma_{pro})$$

It is assumed that a subset of events $\Sigma_{for} \subseteq \Sigma_{act}$ are forcible. An event is said to be forcible if its occurrence can be forced by an external agent or controller to preempt the advance

of time. Each controllable or uncontrollable event can be a forcible event, although we cannot directly prevent the occurrence of an uncontrollable forcible event by disablement.

The controllable event set is defined as

$$\Sigma_c = \Sigma_\tau - \Sigma_{un} = \Sigma_{pro} \cup \{tick\}$$

Due to the nature of time, the tick event can be thought of as a controllable or uncontrollable event. The important point is that only forcible events can preempt the tick event.

If the TDES under consideration is a complex system consisting of several subsystems, it is desirable to be able to build a TDES using an operation on the associated subsystems. The usual synchronous product of TTG`s forces synchronization of the tick transition as it occurs in the component of TTG which may block the advance of time. This is clearly not acceptable. Thus, the required operation, called composition product, is in general different from synchronous product.

Consider two ATGs $G_{1,act}$ and $G_{2,act}$. Let G_{act} be the result of the composition of $G_{1,act}$ and $G_{2,act}$, with the event sets of $\Sigma_{1,act}$ and $\Sigma_{2,act}$. The transition structure of G_{act} is determined by the synchronous product of $G_{1,act}$ and $G_{2,act}$. To obtain the time bounds of G_{act} from the time bounds of its sub-modules, we consider two cases. First, for an event $\sigma \notin \Sigma_{1,act} \cap \Sigma_{2,act}$, the time bounds $[l_\sigma, u_\sigma]$ of σ in G_{act} remain identical to those in the corresponding

ATG model. Second, for $\sigma \in \Sigma_{1,act} \cap \Sigma_{2,act}$, the time bounds of σ in G_{act} are affected by the time bounds of $G_{1,act}$ and $G_{2,act}$.

Let $l_{1,\sigma}$, $u_{1,\sigma}$, $l_{2,\sigma}$, and $u_{2,\sigma}$ be the lower and upper time bounds of σ in $G_{1,act}$ and $G_{2,act}$. Then we may use the following role to obtain l_σ and u_σ, the time bounds of σ in G.

$$(l_\sigma , u_\sigma) = (\max(l_{1,\sigma} , l_{2,\sigma}) , \min(u_{1,\sigma} , u_{2,\sigma})).$$

The above is acceptable if $l_\sigma \leq u_\sigma$.

Otherwise, the composition product is assumed to be undefined.

After these ATGs are combined by the composition operator to build the ATG model of the complex system, the ATG can be converted to TTG.

TTCT introduces the **Comp** procedure to form the composition. Assuming G1act and G2act are ATG models of two subsystems, then their composition product (G_{act}) is:

$$\mathbf{G_{act} = comp(G1_{act} , G2_{act})}$$

To transfer an ATG model to a TTG model, TTCT uses the procedure **Timed-Graph***.

$$\mathbf{G_\tau = Timed\text{-}Graph(G_{act})}$$

G_τ is a TDES (* Note our notation for composition product is different to that used in [12]).

Remark 2.1: Let $G_{1,act}$ and $G_{2,act}$ be two ATG models with TDES models $G_{1,\tau}$ and $G_{2,\tau}$. If the intersection of $\Sigma_{1,act}$ and $\Sigma_{2,act}$ is empty, then the TDES derived from the result of

composition product of $G_{1,act}$ and $G_{2,act}$ will be the same as the synchronous product of $G_{1,\tau}$ and $G_{2,\tau}$.

Modular supervision:

In the analysis and design of modular supervisors, the issues of non-blocking and joint coerciveness must be examined carefully. Joint coerciveness is related to the preemption of the tick by forcible events (the feature of controllability).

A language K is said to be coercive with respect to G if $\forall\ s \in \overline{K}$:

$$\Sigma_K(s) \cap \Sigma_{for} = \Phi \text{ and } tick \in \Sigma_{L(G)}(s) \Rightarrow tick \in \Sigma_K(s)$$

This simply means that tick can be preempted only by forcible events.

Also, languages K1, K2 $\subseteq$ L(G) are jointly coercive with respect to G if K1 $\cap$ K2 is coercive with respect to G.

Supervisory Control With Partial Observations in Timed DES

In this section, the controllability concept is generalized to timed discrete-event systems. The concepts of observability and normality remain the same. The definition of controllability is as follows:

K $\subseteq$ L(G) is said to be controllable with respect to L(G) if $\forall\ s \in \overline{K}$:

$$\begin{cases} \Sigma_{L(G)}(s) \cap \Sigma_{uc} \subseteq \Sigma_K(s) \\ \Sigma_K(s) \cap \Sigma_{for} = \Phi \text{ and } tick \in \Sigma_{L(G)}(s) \Rightarrow tick \in \Sigma_K(s) \end{cases}$$

In the above definition, it is assumed that the tick event can be preempted by forcible events.

The following can be shown [15].

Suppose $K \neq \Phi$ and $K \subseteq L(G)$. The existence of a non-blocking supervisor S is guaranteed such that $L_m(S/G) = K$ if and only if:

i) K is controllable with respect to L(G);

ii) K is observable with respect to L(G); and

iii) K is $L_m(G)$-closed.

Similar to the untimed case, since observability is not closed under union, the concept of normality is used. Thus, one way to synthesize a proper supervisor is to find the supremal element of the normal and controllable sublanguages of the original legal language.

2.3 Fault Diagnosis Systems

In this thesis, a *diagnoser* is an agent used to detect and isolate faults in the system. Considerable research effort has been and is being spent on the design and development of diagnosis systems (see, e.g. [29]). A variety of methodologies, differing both in their theoretical framework and in their design and implementation philosophy, have been proposed.

In our framework in this thesis, we assumed that the plant has some diagnosis systems that can detect and isolate failures with (finite) bounded delay. The underlying mechanism of diagnosis system is not important for us. Nevertheless, we here briefly review some of the fault diagnosis techniques. Our discussion is certainly not exhaustive and the reader is referred to [29] and the references therein for a more complete survey.

Technically, these approaches can be divided into two groups: model-free and model-based approaches.

Model-Free Approaches

Model-free methods are particularly useful in cases where it is hard to obtain a model for the plant. In the following, we briefly discuss two of these techniques.

- Hardware redundancy

In this technique multiple sensors are used to measure a given plant variable. The outputs of sensors measuring the same variable are compared to determine the plant variable and to detect sensor failures.

- Expert systems

In these methods, experience and knowledge of experts is stored as a set of rules and an inference engine is used for failure diagnosis. Some of these approaches are based on

statistical hypothesis testing and signature analysis. In cases in which it is hard to find a model for the plant, these approaches are useful.

Model-Based Techniques

Model-based fault detection methods use a mathematical model of the plant for fault detection. In these methods, the observed behavior of the plant is compared with that expected from the plant model. Based on this comparison an inference is made about the condition of plant (normal or faulty). They can be based on differential and difference equations, e.g., parity relations and unknown input observers [2]. Some of these approaches are described below.

- Fault trees methods

These methods, which have been studied in detail by reliability engineers, provide a pictorial display of the system that can be easily read and understood [23]. These approaches are model-based techniques, and the observed behavior of the plant is compared with that expected from the plant's model. Based on this comparison, an inference is made about the condition (normal or faulty) of the plant.

- Artificial intelligence (AI) model-based reasoning schemes

An example of this type of scheme is the system which was used in NASA's Deep Space 1 spacecraft [2]. It couples the transition system models with the qualitative representation developed in model-based reasoning.

- Discrete event system approaches

These schemes for failure diagnosis are applicable not only to systems that have a discrete nature, but also to systems that traditionally are treated as continuous-variable dynamic systems (such as differential or difference equations) but for diagnosis purpose can be modeled as DES.

In one approach to fault diagnosis, the timed event sequence generated by the DES under supervision is compared to a set of specifications for normal operation called templates. This method is applicable for systems such as high-speed manufacturing lines, in which the plant can be modeled as a set of an unspecified number of timed automata operating parallel to and independent of each other [25].

In some approaches, finite state automata are used for fault diagnosis of discrete event systems. For example, state-based approach is a technique in which it assumed the system state set can be partitioned according to the condition (faulty state) of the system. In this approach, the objective of fault diagnosis is to determine the condition to which the state belonged upon arrival of the last measurement (sensor reading). Both off-line and active and passive on-line diagnoses have been investigated by researchers [24], [28]. In our work on fault recovery, we too use the assumption of the partition of state set according to failure status.

Event-based is another kind of DES approach [6,7,8]. In this scheme, based on observed events, inferences are made regarding the occurrence of unobservable failure events.

In the above approaches, a failure is called diagnosable if it can be detected after a bounded number of events.

Chapter 3

Fault Recovery in Untimed Discrete Event Systems

3.1 Overview

In a control system, component failures could potentially reduce the controller's ability to properly control or observe the plant behavior. Without careful supervision, the behavior of the plant under supervision may no longer satisfy the design specifications. In order to guarantee that the system is operating safely and reliably, the controller should perform fault recovery. This may include replacing faulty components with spare parts or reconfiguration of the control system.

This thesis studies a control structure that provides fault recovery in discrete event systems. In this chapter we focus on fault recovery in systems that can be modeled as untimed discrete event system. Timing issues will be discussed in chapter 4.

Given the complex interactions among components, sub-systems, and processes, system failure is considered to be inevitable and an inherent characteristic of systems. In order to model a failure, it may be added to the DES automaton of the system (plant) as an unobservable event. Once a failure occurs, the plant enters a *faulty-state*.

The event set of the plant includes fault events and possibly recovery events. Fault events are uncontrollable events that represent occurrence of faults in the system. It is assumed that these events are unobservable and therefore cannot be detected by the supervisor. In our framework, we assume the system is equipped with a diagnoser which detects the fault events with some (finite) bounded delay. After the detection of a fault, the diagnoser generates an observable detection event. Upon observing the detection event, the supervisor can make a decision regarding failure recovery.

Recovery events can occur in the recovery mode of the system. These events represent the actions that the system can take to recover from faults. For example, when a command "OPEN VALVE" is sent to a stuck-closed valve, and it does not open, the recovery procedure can be "Replace with redundant valve".

The control problems involved in designing a recovery procedure can be considered as instances of supervisory control problem under partial observation since some of the events, in particular, failures, are unobservable. Thus, the concept of *partial observation* comes into play to help us in the computation of suitable controllers for the system under consideration. We discuss the design of supervisors to meet system's design

specifications for normal operation and recovery from faults. This design may be done based on the RW framework.

3.2 Recovery Framework

Our framework for fault recovery in untimed discrete event systems will be explained in this section. An extension of this approach to timed discrete event systems will be explained in the next chapter.

Recovery procedures typically reconfigure the system. When a fault occurs in a system, it may be possible to shut down the system, or to operate the system at a lower level of conformation to specifications. In some occasions, depending on the fault, these may be the only alternatives. The aim of this approach is to build into the design of the controller, *a priori* knowledge on how to reconfigure itself when failures occurs in real time. To this end, we seek to compute all feasible solutions to control the plant under various faults. For each such solution we synthesize a control logic that satisfies the corresponding specifications. Each of these solutions can be implemented by a controller.

Initially, when the plant starts, a set of controllers is engaged. Upon the detection of a failure in the plant the controller is replaced with a recovery controller corresponding to the particular fault detected. This latter controller is called the *recovery controller.*

3.2.1 System Modes and Control Structure

Suppose the plant can be modeled as a DES and this model describes the system behavior in both normal and faulty situations. We assume that there are p failure modes F_1, ..., F_P. Here, F_i is the i^{th} failure mode and each failure mode corresponds to some kind of failure in a component of the plant or a combination of such failures. We also assume that only one failure event may occur at any time. In other words, there are p+1 conditions (N(normal), F_1, ..., F_P) and that the system can be in only one of them. This is called *single-failure* scenario and it should not be confused with single-failure mode situation in which the system has only one failure mode, i.e., p=1. In addition, we assume that all failure modes are permanent. Fig. 3.1 shows single-failure scenario with permanent failure modes. Recovery failures are shown by dot lines.

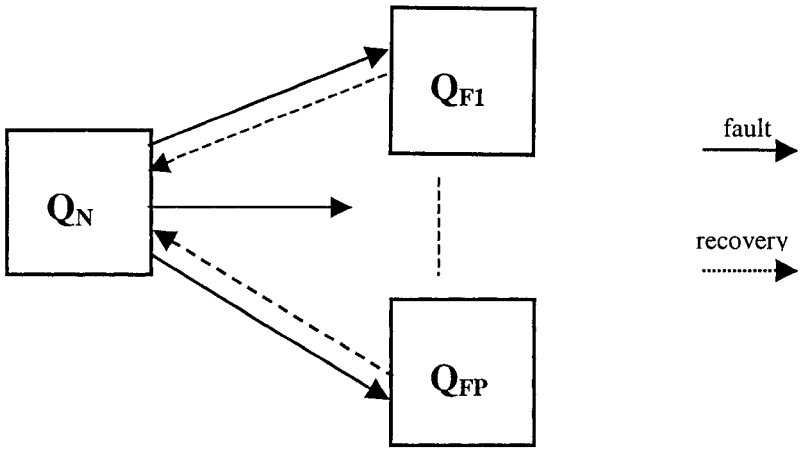


Figure 3.1: Single-failure scenario with permanent failure modes

The condition set of the plant is denoted by $K=\{N, F_1, \ldots, F_P\}$. It is assumed that the state set Q can be partitioned according to the condition of the system: $Q = Q_N \mathbin{\mathring{\cup}} Q_{F1} \mathbin{\mathring{\cup}} \ldots. \mathbin{\mathring{\cup}} Q_{FP}$. Here $\mathring{\cup}$ denotes disjoint union.

In our framework, we assume that a *diagnosis system*, or simply *diagnoser* is available that detects the fault events of DES and reports them to the supervisory controller. The new control system is shown in Fig. 3.2.

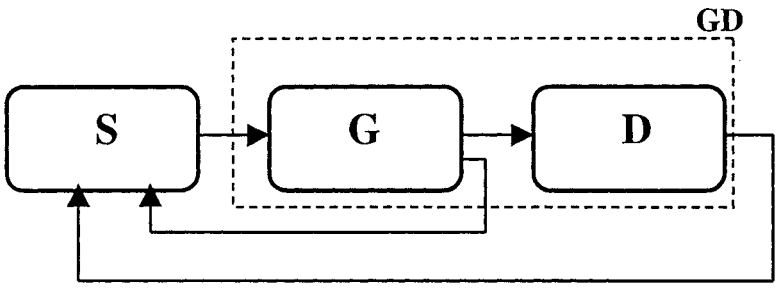


Figure 3.2: Modified supervisory control loop

Here the plant, supervisor, and diagnoser are designated by G, S, and D, respectively. The diagnosis system detects and isolates failure modes with bounded delay; for example, 3 to 5 events (or 3 to 5 clock ticks in timed models). The diagnosis system could be based on any techniques, as long as time bounds for diagnosis delay are available. In general, these time bounds are functions of fault type, plant dynamics and diagnosis techniques.

One of the advantages of our modified supervisory control loop is that it separates supervisory algorithms and diagnosis techniques. By this way, the diagnosis problem and

control problem are almost decoupled. Another advantage is the reduction in the complexity of supervisory control and the recovery procedures (Details will be discussed in the next section).

Of course, in our framework, we need the time bounds for diagnosis delay. Also, our model for diagnoser may be somewhat conservative. For example, let us assume that the system to be controlled has two cycles. In the first cycle, a fault event can be detected within one event delay whereas in the second cycle, detection can be done within three events delay. Our model considers the worst case (three events delay) in both cases.

In contrast, an alternative method is discussed in [8]. There, the authors present an integrated approach to control and diagnosis. They study the active diagnosis problem in the framework of DESs. Since in this case, control and diagnosis problems are integrated, the controller design has, in the worst case, exponential complexity in the number of plant states.

In our modified supervisory control loop, the combination (synchronous product) of the plant and the diagnosis system constitutes the system to be controlled (the system under supervision). This system is denoted by GD in Fig. 3.2.

We can see that the system under supervision has three modes: *normal*, *transient*, and *recovery*.

The **normal mode** describes the system under supervision before the occurrence of a fault in the system. In this situation, the system operates properly and the plant is in its normal mode.

The second mode of the system under supervision is called **transient**. Upon occurrence of faults in the system, the system goes into transient mode. In this mode, the plant is in one of its faulty modes, and the diagnosis system has not detected the fault yet.

In the **recovery mode** of the system under supervision, the plant is in a faulty condition and the diagnosis system has detected the fault.

The time between the occurrence of a fault and its diagnosis is called the *transient time*. The transient time depends on the type of fault, the type of diagnosis system, and the time the failure occurs. This means that the transient time varies from one diagnoser to another and also from one fault to another.

In the RW supervisory control theory, the goal of the supervisory control is to restrict the behavior of the system to a language that satisfies the specifications. In our recovery control problem, since we have three modes for the plant and diagnoser (GD), three sets of specifications can be defined: *normal specifications, transient specifications, and recovery specifications*.

Here, we should mention that transient specifications are typically less restrictive than the normal specifications since as a result of fault, some sensors or actuators are lost.
Recovery specifications describe the required recovery actions. An example of these actions is shutting down the faulty equipment or replacing the faulty equipment with a backup.

As an example, consider temperature control of a power plant. The normal, transient, and recovery specifications could be as follows:

Normal specification: The temperature (T) must be between 80°C and 120°C.

Transient specification: The temperature must be between 60°C and 140°C.

Recovery specification: Shut down the plant while maintaining $50 < T < 150$.

As we can see, temperature lower and upper bounds are more relaxed in transient and recovery modes.

Fig. 3.3 demonstrates the relationship among the modes. It says that if a failure event occurs in the system under supervision, then the system goes from normal mode to the transient mode. In addition, upon detection of a fault or occurrence of the detection event the system enters the recovery mode.

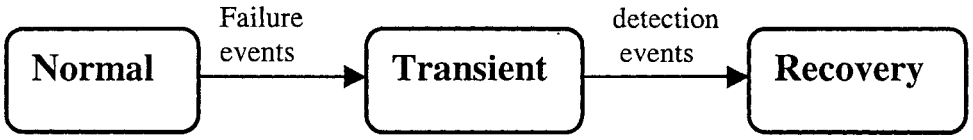


Figure 3.3: The relationship among the modes

In order to restrict the system's behavior to the specifications in different modes, the following controllers (supervisors) have to be computed.

Normal Controller (S_N): A controller for the normal mode based on normal specifications.

Transient Controller (S_T): A controller to enforce transient specifications.

Recovery Controller (S_R): A controller based on recovery specifications.

We consider *modular switching supervisory control* as an effective design alternative to a centralized controller (monolithic supervisor) for the following reasons.

- Modular design is a generally suitable approach for solving complex problems.
- The resultant controller can be more easily updated in case a subtask is modified.
- It is more easily synthesized.
- In our problem, where the number of controllers can be high (depending to the number of faults), a modular approach simplifies the computation of the controllers.

In a modular approach, each controller is assigned to handle a subtask independent of other subtasks and controllers. Therefore, we must make sure that the controllers do not conflict.

Another issue is to ensure smooth transfer from transient mode to recovery mode. This will be addressed in the following section.

3.2.2 Modeling and Problem Formulation

3.2.2.1 Plant Model

Let G denote the plant. It is assumed that this system can be modeled as a finite state automaton defined by a 5-tuple

$$G = (Q, \Sigma_G, \delta, q_0, Q_m),$$

where Σ_G is the set of events, Q is the finite set of states, q_0 is the initial state, Q_m is the set of marked states, and δ: $Q \times \Sigma_G \rightarrow Q$ is the transition partial function. Some of the

events in Σ_G are uncontrollable (Σ_{uc}), i.e., their occurrences cannot be prevented by the controller, while others are controllable (Σ_c). In this regard, Σ_G can be partitioned to Σ_c and Σ_{uc} with $\Sigma_c \cap \Sigma_{uc} = \Phi$. We may also partition Σ_G according to $\Sigma_G = \Sigma_p \cup \Sigma_f \cup \Sigma_r$, where Σ_f and Σ_r are the sets of fault, and recovery events, respectively. The remaining set of events is denoted by Σ_p.

The plant model describes the system behavior in both normal and faulty situations. Suppose there are p failure modes: F_1, ..., F_P. Here, F_i is the i^{th} failure mode and $\Sigma_f = \{f_1, \ldots, f_P\}$. Each failure mode corresponds to some kind of failure in a component of the plant. We assume that the event set Σ_G contains failure events and only that one failure event may occur at any time. In other words, there are p+1 conditions: N(normal), F_1, ..., F_P. This is called *single-failure* scenario and it should not be confused with single-failure mode situation in which the system has only one failure mode, i.e., p=1. In addition, we assume that all failure modes are permanent. Fig. 3.4 shows single-failure scenario with permanent failure modes.

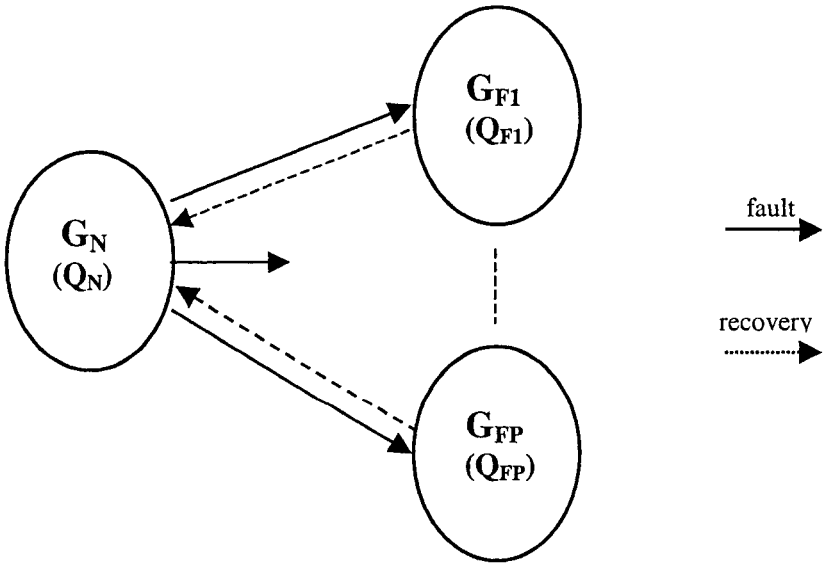


Figure 3.4: Plant G in single-failure scenario with permanent failure modes

Let the condition set of the system be $K=\{N, F_1, \ldots, F_P\}$. Also, with $\overset{\circ}{\cup}$ denoting disjoint union, the state set Q can be partitioned as follows: $Q = Q_N \overset{\circ}{\cup} Q_{F1} \overset{\circ}{\cup} \ldots \overset{\circ}{\cup} Q_{FP}$

The normal mode, denoted by G_N in Fig 3.4, describes the behavior of the system in the normal situation (i.e. the system is functioning properly). Technically speaking, G_N is the sub generator of G that contains Q_N. Similarly $G_{F1}, \ldots, G_{FP}$ describes the plant's behavior in the faulty modes $F_1, \ldots, F_P$.

3.2.2.2 Model for Fault Diagnosis System

In this thesis, a diagnoser is simply an agent that signals the control system when a fault is discovered and indicates the type of the fault. The type of diagnoser and the methodology used by the diagnoser do not play any role in the synthesis of recovery procedures. It is assumed that diagnosis involves a bounded delay.

In our framework, we use an automaton, which we call the diagnostic delay model (*DDM*), to model the diagnoser.

The DDM is defined as a finite state automaton

$$D = (Y, \Sigma_D, \xi, y_0, Y_m)$$

where Y is the state set, y_0 is the initial state, $Y_m \subseteq Y$ is the set of marked states, $\xi : Y \times \Sigma_D \rightarrow Y$ is the transition partial function, and $\Sigma_D = \Sigma_G \cup \Sigma_d$ in the event set including all the plant events and detection events. The detection events represent the

signals sent by the diagnosis system to notify the supervisor of the detection and isolation of failures. The detection events are thus assumed observable.

In the finite detection delay model of the diagnosis system, some events from the event set Σ_G can occur in between the occurrence of a fault and the occurrence of the detection events. In other words, after the occurrence of a fault in the system, the plant can still generate a sequence of events. This continues until the diagnoser detects the fault and generates a detection event. Fig. 3.5 shows the proposed DDM of the diagnoser for untimed DES, assuming single-failure scenario.

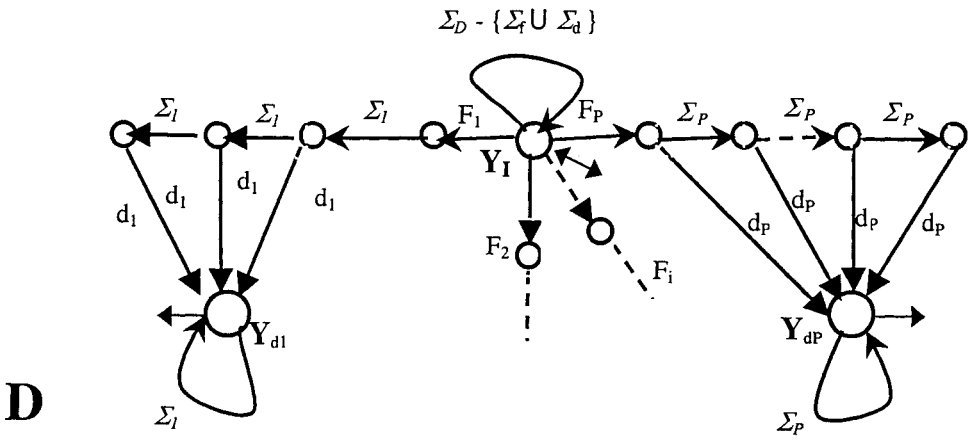


Figure 3.5. Proposed DDM for a diagnoser

In this figure, $\Sigma_i = \Sigma_D - \{d_i\}$ and F_1, F_2, ..., F_P are fault events, and d_1, d_2, ..., d_P are detection events. Here, d_1 is an event signifying the detection and isolation of F_1, and d_2 signifies the detection and isolation of F_2, and so on.

The model indicates that at the initial state Y_I, all members of the event set Σ_G are enabled. This state corresponds to normal condition. If a non-faulty event occurs, the finite state automaton will still remain in the same state (Y_I). On the other hand, if a fault event (e.g. fault F_i) occurs in the system, then the finite state automaton will go to a state corresponding to that fault.

If the diagnoser does not detect the fault before another event occurs, upon the occurrence of the event, the current state of D will change. This repeats until the fault is detected, and the detection event (d_i) is generated by the diagnoser. Occurrence of this last event will take the finite state automaton to state Y_{di}. After this the diagnosis system remains in Y_{di} (unless it is initialized).

It is assumed that a lower and upper bound for the detection delay of each fault is available. For example, in Fig. 3.5, F_1 is diagnosed in between 1 to 3 events.

Modular models can be used for simple representation of a DDM. In order to simplify analysis and implementation of the DDM, dividing the model into modules could be very useful. It is possible to build each of these individual modules and use them concurrently. In other words, the equivalent DDM will be the synchronous product of these modules.

DDM's for two faults with lower and upper bounds of 0 and 3 events are shown in Fig. 3.6.

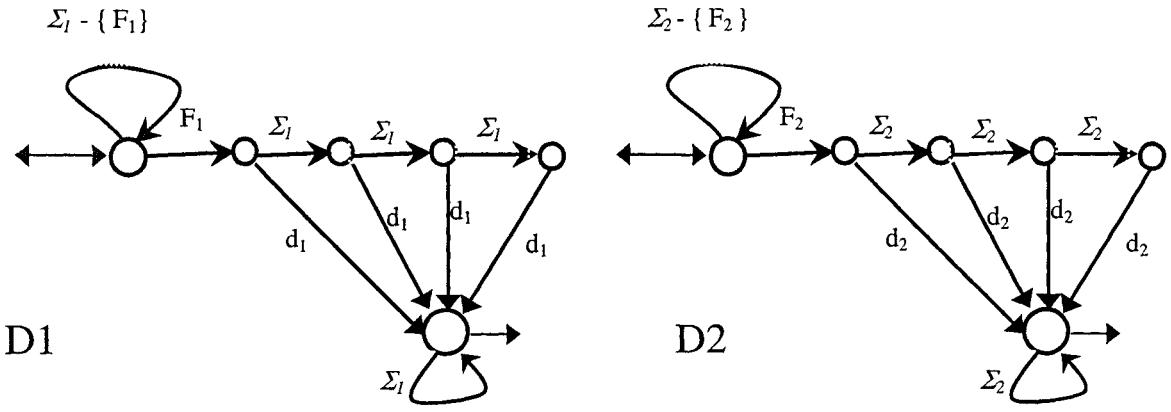


Figure 3.6. DDM's for two faults with lower and upper bounds of 0 and 3 events

3.2.2.3 The System to Be Controlled: GD=sync(G,D)

The system to be controlled (the system under supervision) is the synchronous product of the plant (G) and the diagnosis system (D).

$$\mathbf{GD = sync(G\ ,\ D)}$$

The result is shown in Fig. 3.7.

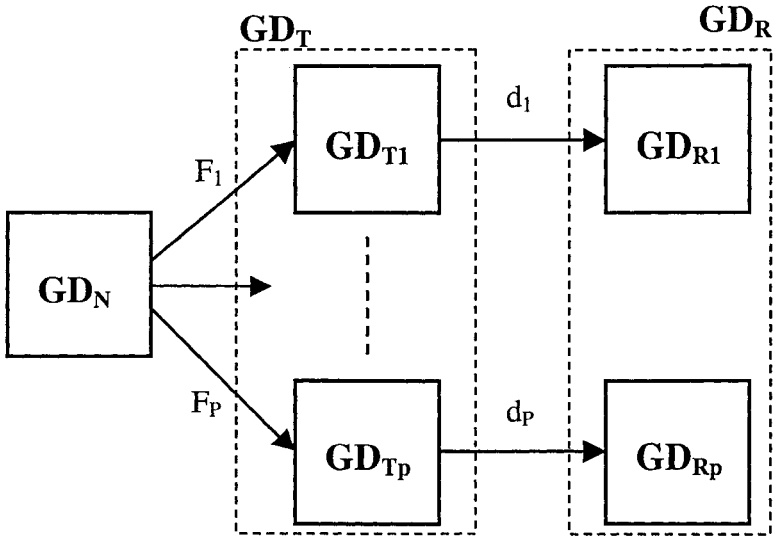


Figure 3.7. The system to be controlled in different modes, GD=**sync**(G,D)

Let $\Sigma = \Sigma_G \cup \Sigma_D = \Sigma_G \cup \Sigma_d$ denote the event set of GD. GD_N is the synchronous product of the plant and the diagnoser in the normal mode (GD_N =**sync**(G_N , D_N)) where G_N includes only the normal condition of the plant G. D_N is illustrated in Fig. 3.8 where Σ_{DN} = $\Sigma_G - (\Sigma_f \cup \Sigma_r)$ (Σ_r is the set of recovery events).

Figure 3.8. Diagnoser in the normal mode

GD_{NT1}, ..., GD_{NTP} describe GD in the transient modes corresponding to F_1, ..., F_P, respectively. Moreover, GD_{NT} denotes GD in normal and transient modes. Also, GD_{NTR} (=GD) describes the system in all normal, transient, and recovery modes.

3.2.2.4 Specification Models

In general, for each mode (normal, transient, and recovery), the system has a set of specifications. We call the specifications of the normal, transient, and recovery modes *normal specifications* (E_N), *transient specifications* (E_T), and *recovery specifications* (E_R), respectively.

To represent the specifications, we construct suitable automata. We also use the **meet** procedure of TTCT to calculate the intersection of the specifications in each mode.

E_N is constructed based on the normal specifications that the plant under supervision should be confined to. It does not include the fault, detection, and recovery events.

E_T is defined for the transient time. The transient specifications are defined in order to control the behavior of the system as much as possible. Transient specifications are typically less restrictive than normal specifications; E_T may allow the system to work with a degraded functionality. That is because some components of the system have become faulty and do not function properly in the transient mode.

E_R is used in computing recovery control actions for faults. It may contain some recipes for shutting down the faulty system, repairing a subsystem, or replacing an instrument with its backup.

3.2.2.5 Example 3.1: Transfer Line

An industrial transfer line consists of two *machines* (**M1** and **M2**) linked by a *buffer* (**B**). The configuration of the transfer line is shown in Fig. 3.9.

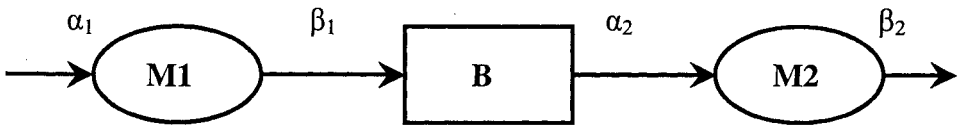


Figure 3.9: Example 3.1. Transfer line

Machine 1, processes workpieces and deposits them into the buffer. Machine 2 removes workpieces from the buffer and processes the workpieces. The DES's representing the machines are displayed in Fig. 3.10.

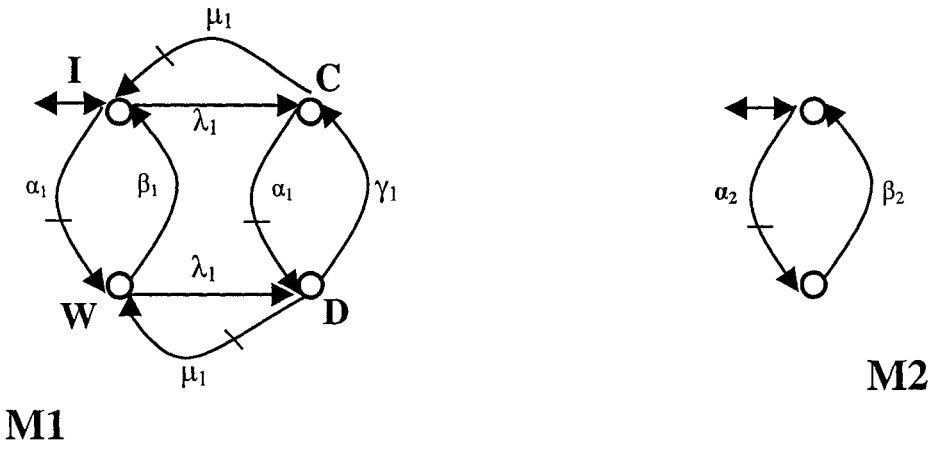


Figure 3.10: Example 3.1. DES models of machines 1 and 2

Let us assume that **M1** has a sensor that signals machine's deposit operation (event β_1). If this sensor becomes faulty (event λ_1), it stops sending signal about deposit operation (event γ_1 signifies *workpiece deposit without sensor information*). So we assume that all events in the transfer line are observable except λ_1 and γ_1.

The transfer line operates as follows: Initially the buffer is empty. **M1** either becomes faulty and enters **C** (event λ_1), or takes a workpiece (event α_1) from an infinite input bin and enters **W**. Subsequently, **M1** either becomes faulty and enters **D** (event λ_1), or successfully completes its work cycle, deposits the workpiece in the buffer, and returns to **I** (event β_1). In faulty state C, **M1** either can take another workpiece and enters D (event α_1), or gets repaired and returns to I (event μ_1). In D, **M1** completes it processing operation in a faulty mode and enters C (event γ_1), or gets repaired and returns to **W** (event μ_1).

M2, takes its workpiece from the buffer (event α_2) and deposits it when finished, in an output bin (event β_2). We assume that **M2** always works properly.

The **specifications** for admissible operation are:

Normal specifications:

1. The buffer must not underflow.

2. There should be no more than one workpiece in the buffer (buffer capacity for the normal mode = 1).

3. Recovery event must be disabled in normal mode.

4. The system under supervision must be non-blocking.

Transient specifications:

1. The buffer must not underflow.

2. There should be no more than two workpieces in the buffer (buffer capacity for the transient mode = 2).

3. Recovery event may not be used in transient mode.

Recovery specifications:

1. Machine 1 should not take new workpieces.

2. There should be no more than three workpieces in the buffer (buffer capacity for the recovery mode = 3)

3. Machine 2 should finish the work on workpieces known to be in the buffer.

4. The system under supervision must be non-blocking.

The plant of the transfer line is

$$\mathbf{G = sync(M1, M2)}$$

In the normal mode, machines 1 and 2 are:

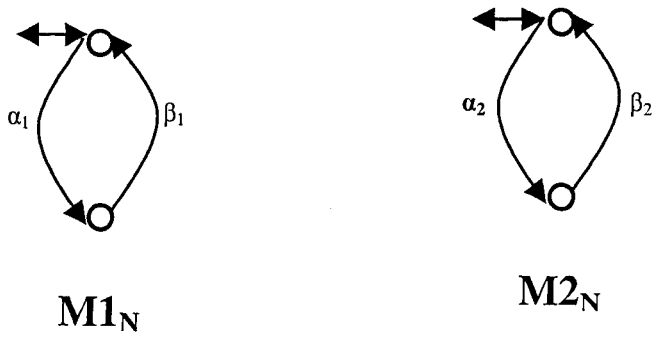


Figure 3.11: Example 3.1. Normal models of machines 1 and 2

In the Fig. 3.11, it can be seen that events λ_1, γ_1 and μ_1, which are related to the faulty condition of M1, are deleted from the original structure of the machine. In other words, the events that illustrate the behavior of the machine after fault occurs are not part of the normal mode of the system.

To construct the normal DES model of the transfer line, the synchronous product (TTCT **sync** procedure) is used.

$$\mathbf{IG_N = sync(\ M1_N\ ,\ M2_N\ )}$$

The result of this procedure is shown in Fig. 3.12.

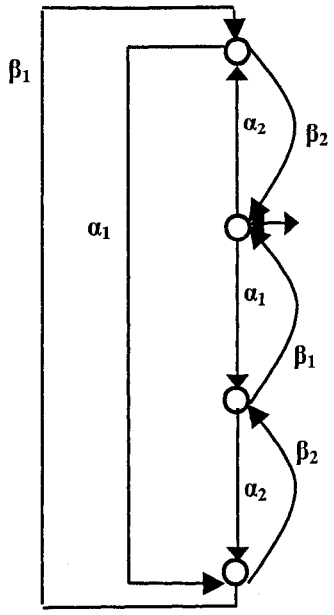


Figure 3.12: Example 3.1. Normal model of the transfer line

As explained in the following section, to design supervisors to enforce transient specifications, system's model in normal-transient and normal-transient-recovery modes (GD_{NT} and GD_{NTR} in Fig. 3.20) are required. In this example, models can be computed as explained in the following.

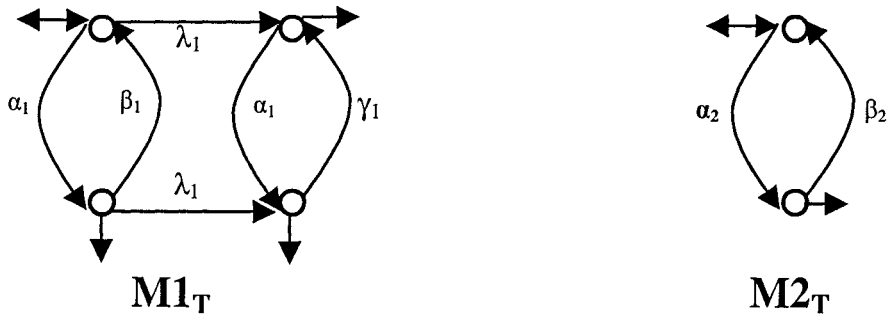


Figure 3.13: Example 3.1. Transient models of machines 1 and 2

Let $\mathbf{M1_T}$ and $\mathbf{M2_T}$ be the machines 1 and 2 in Fig. 3.13. Now let

$$\mathbf{IG_{NT} = sync(M1_T , M2_T)}$$

Note that all states in IG_{NT} are marked. This is because non-blocking property is not among the transient specifications. This issue will be discussed in more detail later.

Consider $\mathbf{M1_R}$ and $\mathbf{M2_R}$ given in Fig. 3.14.

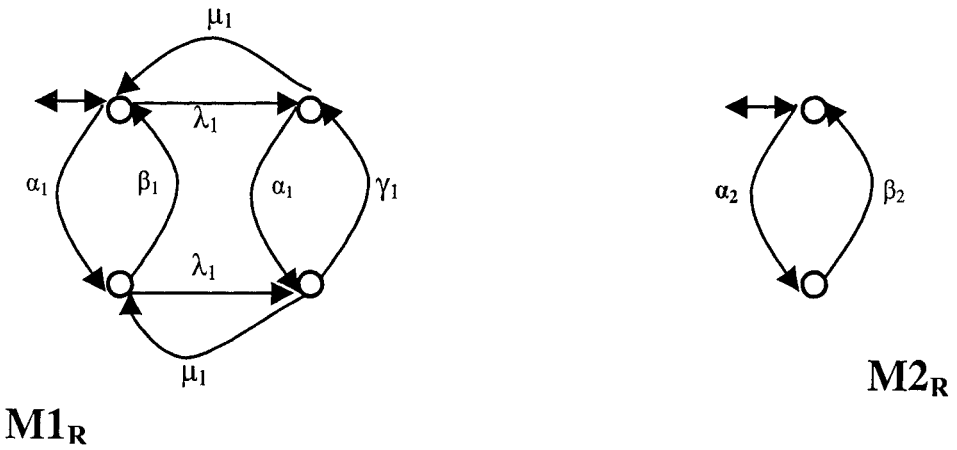


Figure 3.14: Example 3.1. Recovery models of machines 1 and 2

It can be seen that these models are the same as the original structure of these machines. Using synchronous product, the complete model of the small factory can be constructed as follows.

$$\mathbf{IG_{NTR} = sync(M1_R, M2_R)}$$

Now, let us construct the model of the diagnoser assuming the detection delay is between one and two events. The DDM for a minimum delay of one event and a maximum delay of two events is shown in Fig. 3.15.

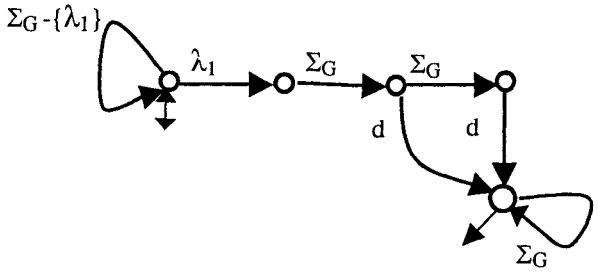


Figure 3.15: Example 3.1. DDM with delay between 1 and 2 ($\mathbf{D_R}$)

where $\Sigma_G = \{\alpha_1, \alpha_2, \beta_1, \beta_2, \lambda_1, \gamma_1, \mu_1\}$.

Let $\Sigma_{G,r} = \Sigma_G - \{\mu_1\}$. The DDM of the diagnoser used in the calculation of the normal-transient mode is displayed in Fig. 3.16.

It should be noted that recovery events are control actions used in the recovery mode only. Therefore, we do not consider them in the normal and transient models.

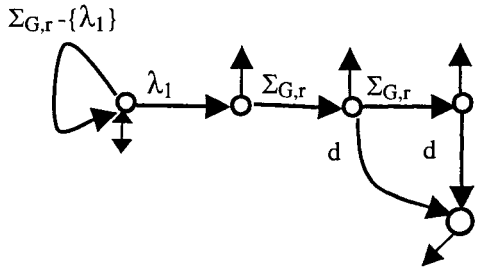


Figure 3.16: Example 3.1. DDM with delay between 1 and 2 for the transient mode ($\mathbf{D_T}$)

The system to be controlled in each mode is as follows:

Normal mode: $\mathbf{IGD_N = IG_N}$

Normal-Transient mode: $\mathbf{IGD_{NT} = sync(IG_{NT}, D_T)}$

Normal-Transient-Recovery mode: $\mathbf{IGD_{NTR} = sync(IG_{NTR}, D_R)}$

The normal, transient, and recovery specifications are built as follows:

Normal specifications ($\mathbf{IE_N}$):

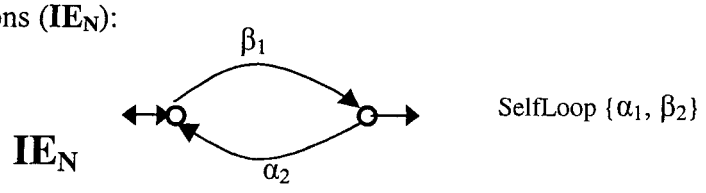


Figure 3.17: Example 3.1. Normal specifications for the transfer line

Transient specifications ($\mathbf{IE_T}$):

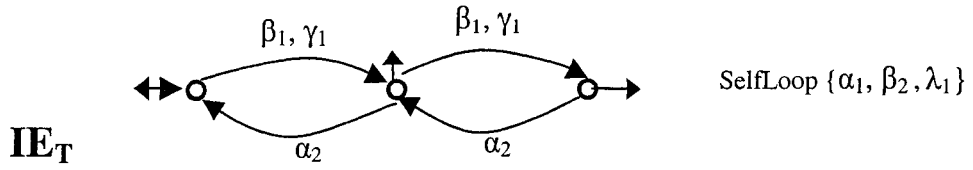


Figure 3.18: Example 3.1. Transient specifications for transfer line

Recovery specifications ($\mathbf{IE_R}$):

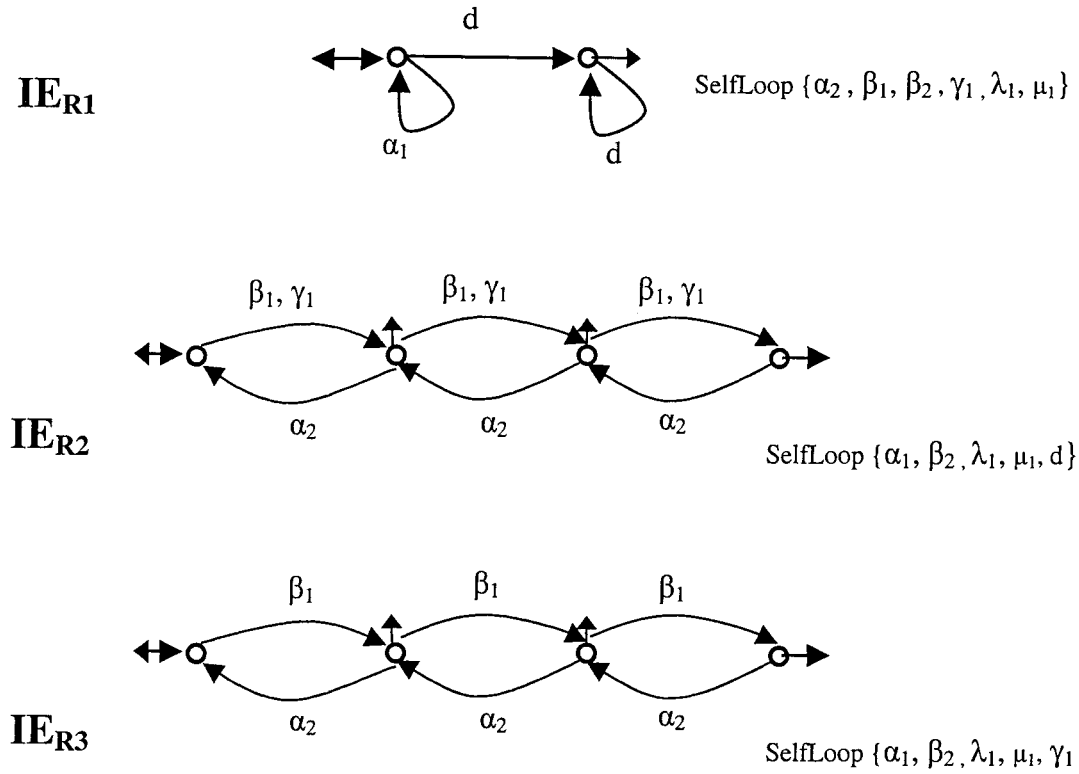


Figure 3.19: Example 3.1. Recovery specifications for transfer line

Combining the recovery specification languages we find the equivalent recovery specification.

$$\mathbf{IE_R = meet(\ IE_{R1}\ ,\ IE_{R2}\ ,\ IE_{R3})}$$

3.2.3 Recovery Procedure and Supervisor Design

This section describes the design procedure proposed in this thesis. First, we consider the case of single permanent fault (p=1), and then in the next step, the procedure will be explained for multiple failures.

3.2.3.1 Single Failure (p=1)

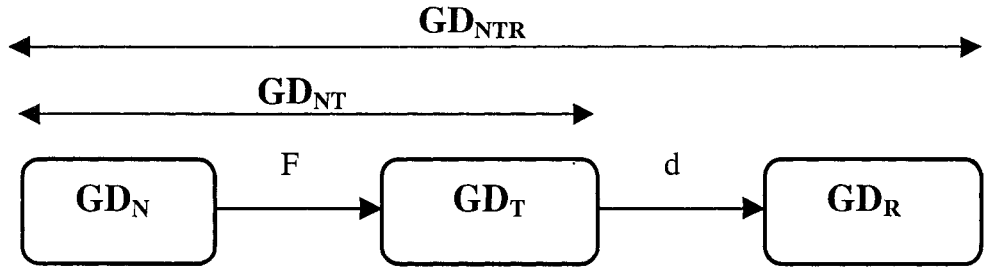


Figure 3.20: GD in different modes for single failure case

Fig. 3.20 illustrates the plant under supervision in different modes. Using this figure, we bring in the following definitions:

GD_N (which is isomorph to G_N) represents the system to be controlled in the normal mode. GD_{NT} is the system containing plant and diagnoser in both normal and transient modes. GD_{NTR} represents the entire plant from the normal to the recovery mode. The combined model (synchronous product) of the system containing the plant and the DDM of the diagnoser in transient and recovery modes are designated by GD_T and GD_R.

E_N and S_N represent the normal specifications and supervisor for the normal mode, respectively. The transient specifications and supervisor are denoted by E_T and S_T. S_{NT} denotes the synchronous product of the normal and transient supervisors (S_{NT}=**sync**(S_N,S_T)).

E_R and S_R represent the recovery specifications and recovery supervisor synthesized for the faulty mode F.

The event set Σ_{GD} is defined to be $\Sigma_{GD} = \Sigma_G \cup \Sigma_D = \Sigma_P \cup \Sigma_f \cup \Sigma_d \cup \Sigma_r$.

Here, Σ_r is the set of recovery events and Σ_P is the set of typical plant events, excluding fault, detection, and recovery events.

Now, we are ready to compute the required controllers for the purpose of fault recovery.

Table 3.1 contains a list of automata that are constructed in order to compute the controllers for the case of single failure. It also contains the event sets over which the DESs are defined.

Table 3.1. Useful automata for recovery procedure in single failure case

Plant	Specification	Supervisor	Event Set
GD_N	E_N	S_N	Σ_p
GD_{NT}	E_T	S_T	Σ_p +{F,d}
GD_{NTR}	E_R	S_R	Σ_p +{F,d,r}
-	-	S_{NT}	Σ_p +{F,d}

Note that the design of S_N, S_T, and S_R can be based on the RW theory or any other method. In addition, in designing S_T, all states of GD_{NT} are marked since blocking is not considered in transient mode.

The diagram of the recovery procedure at implementation time is shown in Fig. 3.21. This diagram illustrates the plant model and the controllers used in different modes of the system.

At the beginning of the implementation, in the normal mode of the system, the conjunction of the S_{NT} and S_R controls the plant in a supervisory feedback loop. S_R is involved with the system from the beginning to ensure that later, in case of failure, recovery specifications can be met. In many cases, S_{NT} is more restrictive than S_R and it controls the plant directly, and S_R follows the event sequences generated by the plant under the supervision of S_{NT}.

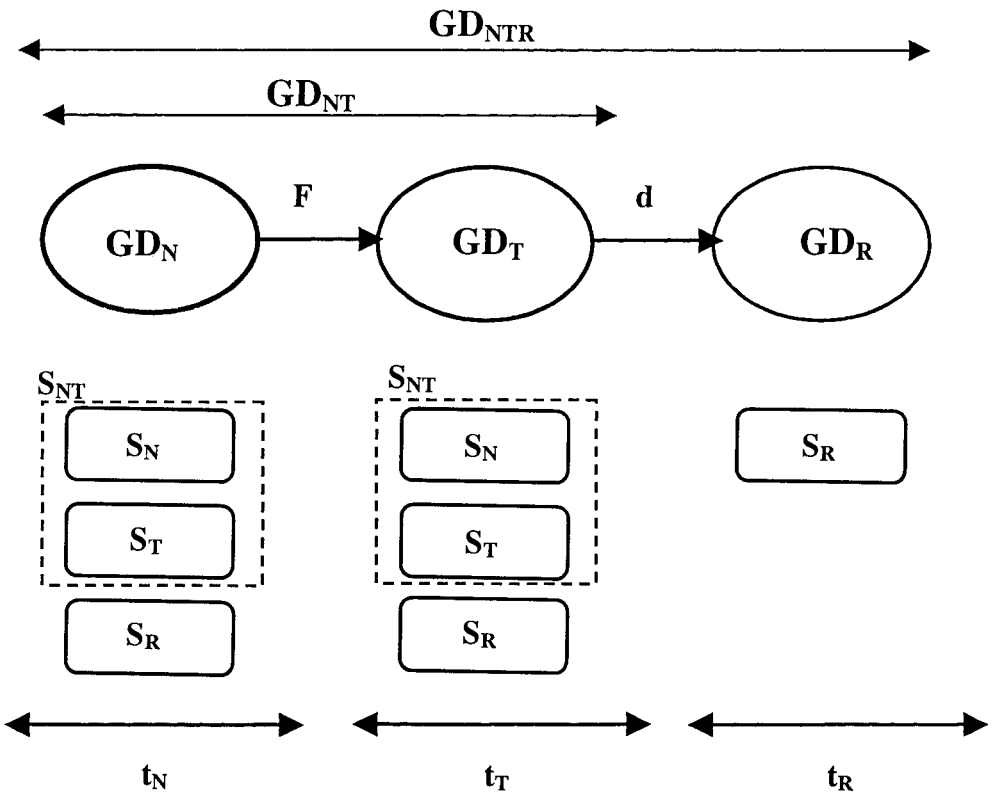


Figure 3.21. The diagram of the recovery procedure in the single failure case

If a fault occurs, the system enters the transient mode. During the transient time S_{NT} (or S_{NTR}=**meet**(S_{NT} , S_R)) continues its supervision until the diagnoser detects the fault and generates the related detection event (d). Upon occurrence of the detection event, S_{NT} is disabled and the recovery supervisor (S_R) continues to control the system alone.

As mentioned above the synchronous product of S_N and S_T (i.e. S_{NT}) is used in this procedure, and we display S_N and S_T separately in the timed diagram for clarity.

In the normal or transient modes, S_R usually only follows the events happening in the plant. In some cases, it may have to restrict plant behavior in normal or transient modes so that E_R can later be met during recovery. This will be discussed in more detail in section 3.2.4.

3.2.3.2 Multiple Failures

In order to discuss the recovery procedure for the multiple failures, we use the same model displayed in Fig. 3.7.

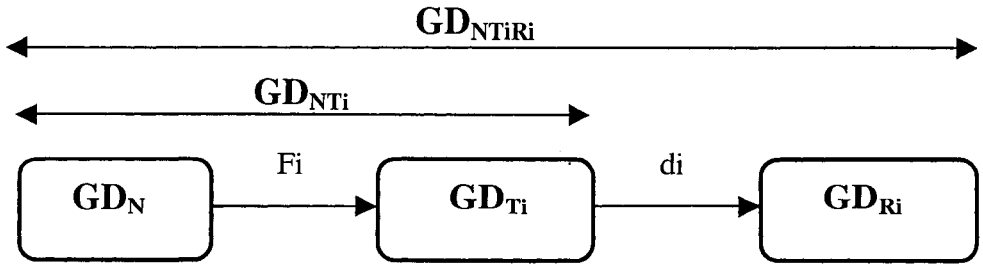


Figure 3.22: GD in different modes for multiple failures case

As before, GD_N, which is isomorph to G_N, represents the system to be controlled in the normal mode. GD_{NT} models the system (plant and diagnoser) in both normal and transient mode.

In Fig. 3.22, GD_{NTiRi} represents the entire plant from normal to transient and to recovery in the case of fault i. S_T is the transient supervisor that is constructed to enforce the transient specifications (E_T). E_{Ri} and S_{Ri} denote the recovery specification and controller for fault i. The conjunction of S_{Ri}'s is called S_R, or S_R = **meet**(S_{R1}, ..., S_{RP}).

Table 3.2 lists the useful automata that should be constructed. It also gives the event sets based on which each supervisor should be designed.

Table 3.2. Useful automata for recovery procedure in the case of multiple failures

Plant	Specification	Supervisor	Event Set
GD_N	E_N	S_N	Σ_p
GD_{NT}	E_T	S_T	$\Sigma_p \cup \Sigma_f \cup \Sigma_d$
GD_{NTiRi}	E_{Ri}	S_{Ri}	$\Sigma_p \cup \Sigma_f \cup \Sigma_d \cup \Sigma_r$
-	-	S_{NT}	$\Sigma_p \cup \Sigma_f \cup \Sigma_d$

Similar to the case of single failure, the conjunction of the supervisors (**meet** (S_{NT}, S_R)) is in the feedback loop.

After the detection of fault i, S_{NT} and p-1 recovery supervisors (S_{Rj}, for all $j \neq i$) stop controlling the system (i.e., removed from the feedback loop) and only the recovery supervisor corresponding to fault i (S_{Ri}) will control the system. Fig. 3.23 shows the diagram of the recovery procedure in the case of multiple failures.

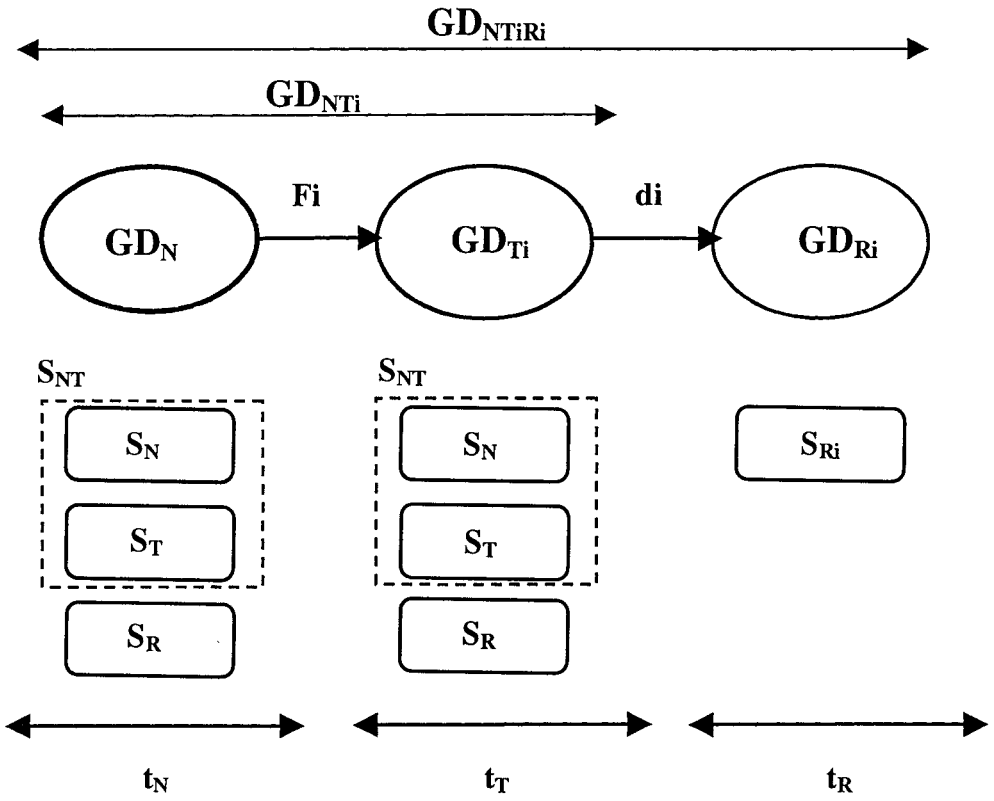


Figure 3.23. The diagram of the recovery procedure for the case of multiple failures

Remark 3.1: Before the occurrence of a fault in the system, the normal supervisor controls the plant; then, until detection of the fault, the transient supervisor should control the plant. Because faults are unobservable, S_N, S_T (thus, S_{NT}=**sync**(S_N, S_T)), and S_R will be in feedback with the plant in both normal and transient modes.

3.2.3.3 Example 3.1 (cont'd)

Now, we design supervisors for the transfer line using the RW framework. We let the plant determine the marking of the states. Therefore, in this thesis, we mark all of the states of the supervisors. In other words, the closed and marked behaviors of DES models of these supervisors are the same.

Normal Mode

For the transfer line, the normal supervisor can be computed as:

$$IS_N = \mathbf{supcon}(IGD_N, IE_N)$$

IS_N

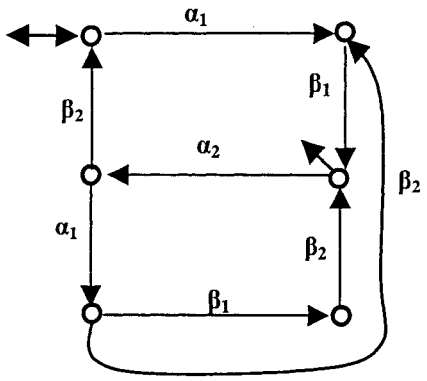


Figure 3.24: Example 3.1. Normal supervisor

Note that in this example, the events in the normal mode are observable. If some of those events are unobservable, the theory of the RW supervisory control problem under partial observation can be used to compute an admissible supervisor for the system.

Transient and Recovery Modes

Since the transient and recovery modes include fault events, and since these events are unobservable, designing S_T and S_R can be considered as cases of supervisory control problem under partial observation. One way to compute the supervisors is to use the framework of Lin-Wonham (LW) discussed in chapter 2. This method is a step-by-step design that can be implemented using TTCT procedures.

The transient and recovery supervisors, using LW procedure, are computed as follows.

Transient Supervisor (IS_T)

Since we do not need to investigate the non-blocking property for this mode, we mark all of the states of $\mathbf{IGD_{NT}}$. Therefore, we eliminate step 7 (**nonconflict**(IGD_{NT} , IS_T) = true ?) from the LW procedure.

1 **ITN = supnorm(IE_T , IGD_{NT} , NULL)** , **NULL=** $\{\lambda_1, \gamma_1\}$
2 **ITNO = project(ITN , NULL)**
3 **ITGO = project(IGD_{NT} , NULL)**
4 **ITKO = supcon(ITGO , ITNO)**
5 **ITKODAT = condat(ITGO , ITKO)**
6 **IS_T = selfloop(ITKO , NULL)**
7 **ITK = meet(IGD_{NT} , IS_T)**
8 **ITK** nonempty **?**

As expected the result of condat procedure (Fig. 3.25) shows that no uncontrollable event is disabled by IS_T. The answer to step 8 is "yes". Thus, IS_T is an admissible supervisor for IGD_{NT}.

```
ITKODAT
Control data are displayed as a list of supervisor states
where disabling occurs, together with the events that must
be disabled or forced there.

control data:
     0:   5                              1:   5
     7:   1                              9:   5
    14:   1                             15:   1
```

Figure 3.25: Example 3.1. Results of **condat** procedure

The supervisor S_{NT} is the synchronous product of S_N and S_T or

$$\mathbf{IS_{NT} = sync(\ IS_N\ ,\ IS_T)}$$

Recovery Supervisor (IS_R)

In order to compute the recovery supervisor (IS_R), following a modular approach, we intuitively design two supervisors for the first and second recovery specifications, and use the LW method for designing the third supervisor based on the third specification. IS_R will be the conjunction of these supervisors.

In order to satisfy the first and second recovery specifications, it can be verified that the proper supervisors for IGD_{NTR}, are the following:

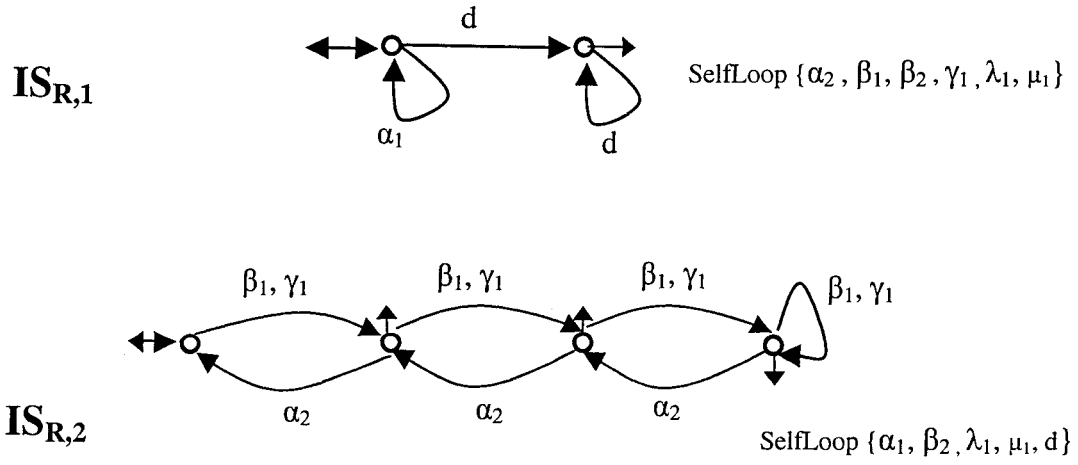


Figure 3.26: Example 3.1. Supervisors for first and second specifications

To compute the third recovery supervisor ($IS_{R,3}$), we use the LW method:

1 **IRN = supnorm(IE_{R3} , IGD_{NTR} , NULL)** , **NULL= $\{\lambda_1, \gamma_1\}$**

2 **IRNO = project(IRN , NULL)**

3 **IRGO = project(IGD_{NTR} , NULL)**

4 **IRKO = supcon(IRGO , IRNO)**

5 **IRKODAT = condat(IRGO , IRKO)**

6 **$IS_{R,3}$ = selfloop(IRKO , NULL)**

7 **nonconflict(IGD_{NTR} , $IS_{R,3}$)** = true **?**

8 **IRK = meet(IGD_{NTR} , $IS_{R,3}$)**

9 **IRK** nonempty **?**

The answers to questions 7 and 9 are "yes". The obtained recovery supervisor, $IS_{R,3}$, has 88 states and 413 transitions. The result of step 5, which is given in Fig. 3.27, illustrates that $IS_{R,3}$ is an admissible supervisor for IGD_{NTR}. $IS_{R,3}$ is a non-blocking supervisor for IGD_{NTR}.

```
IRkODAT
Control data are displayed as a list of supervisor states
where disabling occurs, together with the events that must
be disabled or forced there.

control data:
     0:   5                              1:   5
     2:   5                              5:   5
     6:   5                              8:   5
    13:   5                             14:   5
    25:   5                             39:   5
    41:   1                             53:   1
    66:   3                             68:   1
    76:   1                             83:   1
    84:   1                             86:   1
```

Figure 3.27: Example 3.1. Results of **condat** procedure

IS_R is computed as follows:

$$\mathbf{IS_R = meet\ (IS_{R,1}, IS_{R,2}, IS_{R,3})}$$

Remark 3.2: In the transfer line example, the detection delay of the diagnoser was, at most, two events. In practice, this delay has lower and upper bounds. If the upper bound is too high, then the design procedure will not be able to produce an answer (in fact, the supervisory control problem may not have a solution). The design procedure may be repeated for increasing values of diagnosis delay, to find the maximum value for which a solution is available.

3.2.4 Decoupling Condition

In the normal and transient modes, the conjunction of S_{NT} and S_R is controlling the plant. S_R may have to disable some events to ensure recovery specifications will be satisfied later.

In some cases, S_R simply follows the plant throughout the normal and transient modes. Therefore, S_{NT} and S_R are effectively decoupled in time: S_{NT} influences the plant during the normal and transient modes and S_R during the recovery mode. Decoupling occurs if and only if

$$\mathbf{L}\,(S_{NT}/GD_{NT}) \subseteq \mathbf{L}\,(S_{Ri}/GD_{NT})$$

We call this condition **the decoupling condition.**

The above decoupling condition means that the portion of the closed language of the transient plant under supervision of the normal-transient supervisor, which is generated before the controller switches to recovery mode, should be a subset of the closed language that would have been generated by the transient plant under the supervision of the recovery supervisor.

Example 3.1 (Cont'd): To verify the decoupling condition for the transfer line, we must examine:

$$\mathbf{L(\ IS_{NT}/IGD_{NT}) \subseteq L(\ IS_R/IGD_{NT})}$$

Since all of the states of IS_{NT}, IS_R, and IGD_{NT} are marked, their closed and marked behaviors are equal. Therefore, using TTCT procedures, the verification procedure is as follows:

1. ISRIGDNTCO = complement(IS_R/IGD_{NT}, --)

2. trim(meet(IS_{NT}/IGD_{NT} , ISRIGDNTCO)) = EMPTY ?

The answer to step 2 is "yes", and this means that the decoupling condition is satisfied by IS_R and IS_{NT} for the transfer line.

3.2.5 The Issue of Non-Blocking

In problems in which blocking properties are studied, we would like the plant under supervision to be non-blocking during its normal operation and in case of failure, be non-blocking during the recovery procedures.

Let us assume that S_N is non-blocking for GD_N (i.e., S_N/GD_N is non-blocking), and S_{Ri} is non-blocking for GD_{NTiRi} (i=1, ..., P) (i.e., S_{Ri}/GD_{NTiRi} is non-blocking).

In the **normal mode**, the plant under supervision **meet**$(S_{NT},S_R)/GD_N$ should be non-blocking, or

nonconflict(GD_N , meet(S_{NT}, S_R)) should be true. (*)

This is a modular supervisory control problem. It can be shown that [11] if S_{Ri}'s and S_{NT} are admissible and non-blocking supervisors for GD_N, and $L_m(S_{NT}/GD_N)$ and $L_m(S_{Ri}/GD_N)$'s are nonconflicting, then **meet**$(S_{NT}, S_R)/GD_N$ will be non-blocking. This means that the normal plant under supervision of the equivalent supervisor **meet**(S_{NT}, S_R) will be non-blocking.

If the decoupling condition is satisfied then the equivalent supervisor in the normal mode is S_{NT}. In this case, to verify the non-blocking property, we can see if the following holds:

$$\textbf{nonconflict}(GD_N, S_{NT}) = \text{true}$$

In the **transient mode**, non-blocking is not part of the specifications of the plant under supervision. This is reasonable until the fault is detected and a recovery supervisor takes sole control of the plant.

In the **recovery mode**, only one of the recovery supervisors is in the feedback loop. In the normal and transient modes, however, the conjunction (**meet**) of the supervisors (S_{NT} and S_{Ri}'s) is in control. Therefore, the language generated by the plant under the equivalent supervisor is a sublanguage of the plant under the recovery supervisor or

$$\mathbf{L}(\mathbf{meet}(S_{NT}, S_R)/GD_{NT}) \subseteq \mathbf{L}(S_{Ri}/GD_{NT}) \qquad i = 1, \ldots, p \qquad (**)$$

Let us formulate the non-blocking condition for the recovery mode. We start with the case of single failure shown in Fig. 3.20.

First we construct a supervisor $\tilde{S}_{NT}$ that behaves similar to S_{NT} in the normal and transient modes and that becomes idle during the recovery mode. We build $\tilde{S}_{NT}$ by adding a dump state (X_d) to all states of S_{NT}, and mark X_d. We also attach a transition (X_d, α , X_d) for each event $\alpha \in \Sigma$ at this state. In addition, we remove all d transitions in S_{NT} and add a transition (q , d , X_d) from every state q in S_{NT} to the dump state (X_d) (Fig. 3.28). If coupled with the system GD, $\tilde{S}_{NT}$ would behave similar to S_{NT} during the normal and transient modes and after failure detection (after it enters X_d), it would stop disabling events and effectively it would be out of the control loop.

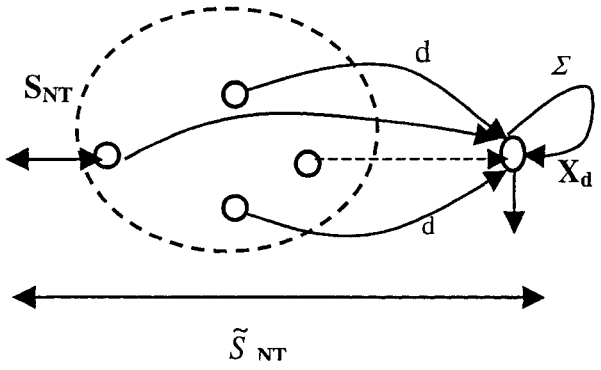


Figure 3.28: $\tilde{S}_{NT}$

Let S_{NTR} be S_{NTR} = **meet**($\tilde{S}_{NT}$, S_R). We can think of S_{NTR} as the supervisor that controls the system under supervision for the entire time from normal more to recovery.

Now, the non-blocking condition for the recovery mode will be as follows:

S_{NTR}/GD_{NTR} is non-blocking, if and only if

$$\overline{L}_m(S_{NTR}/GD_{NTR}) = L(S_{NTR}/GD_{NTR})$$

Theorem 3.1 Suppose S_R is a non-blocking supervisor for GD_{NTR}. If **meet**(S_{NT}, S_R) is a non-blocking supervisor for GD_N, then S_{NTR} is a non-blocking supervisor for GD_{NTR}.

Proof: The sequences in $L(S_{NTR}/GD_{NTR})$ can be partitioned into three groups:

(a) The sequences that keep GD in the normal mode and thus do not contain failure or detection events.

(b) The sequences that take GD to the transient mode and thus contain at least one failure event and have no detection event.

(c) The sequences that take GD to the recovery mode and therefore contain at least one detection event.

In case (a), since by assumption S_{NTR}/GD_N is non-blocking, from any normal state in S_{NTR}/GD_{NTR} there is a path in the normal mode to some marked states in the normal mode.

In case (c), from equation (**) (page: 69) we conclude that any sequence $s= \sigma_1, \ldots, \sigma_m d$ ($\sigma_i \neq d$) in $L(S_{NTR}/GD_{NTR})$ that takes GD_{NTR} to some state x and S_R to some state y, will also take GD_{NTR} to x and S_R to y in S_R/GD_{NTR}. Note that x is the state immediately after detection. Since S_R/GD_{NTR} is non-blocking, there is a sequence of events s' to take S_R/GD_{NTR} to a marked state. After detection the action of S_R and S_{NTR} are identical. Thus, the same sequence takes S_{NTR}/GD_{NTR} to the same marked state. In other words, $ss' \in L_m(S_{NTR}/GD_{NTR})$.

It can be seen that the aforementioned argument applies to any $s = \sigma_1, \ldots, \sigma_m d s_1$ ($\sigma_i \neq d$).

In case (b), for any sequence $s \in L(S_{NTR}/GD_{NTR})$ that includes a failure f but does not includes d , there exist s' such that $ss'd \in L(S_{NTR}/GD_{NTR})$ (because the diagnosis system detects the failure with (finite) bounded delay). By (c), there exist s" such that $ss'ds" \in L_m(S_{NTR}/GD_{NTR})$.

Therefore, $L(S_{NTR}/GD_{NTR}) \subseteq \overline{L}_m(S_{NTR}/GD_{NTR})$. Since $\overline{L}_m(S_{NTR}/GD_{NTR}) \subseteq L(S_{NTR}/GD_{NTR})$ is always true, we conclude that

$$\overline{L}_m(S_{NTR}/GD_{NTR}) = L(S_{NTR}/GD_{NTR}).$$

In other words, S_{NTR} is a non-blocking supervisor for GD_{NTR}. □

For multiple failures, we define S_{NTRi} as follows:

$$\mathbf{S_{NTRi} = meet(meet(\tilde{S}_{NT}, \tilde{S}_{R1}, \ldots, \tilde{S}_{Ri-1}, \tilde{S}_{Ri+1}, \ldots, \tilde{S}_{RP}), S_{Ri})}$$

Here, $\tilde{S}_{NT}$, $\tilde{S}_{R1}$, …, $\tilde{S}_{Ri-1}$, $\tilde{S}_{Ri+1}$, …, $\tilde{S}_{RP}$ are respectively extensions of S_{NT}, S_{R1}, …, S_{Ri-1}, S_{Ri+1}, …, S_{RP}, and must be calculated using the same procedure explained above for obtaining $\tilde{S}_{NT}$.

We extend the non-blocking condition for the recovery mode corresponding to a fault f_i as follows:

S_{NTRi}/GD_{NTiRi} is non-blocking, if and only if

$$\overline{L}_m(S_{NTRi}/GD_{NTiRi}) = L(S_{NTRi}/GD_{NTiRi})$$

The following theorem is an extension of Theorem 3.1.

Theorem 3.2 Suppose S_{Ri} is a non-blocking supervisor for GD_{NtiRi}, with $1 \leq i \leq p$. If **meet**(S_{NT}, S_R) is a non-blocking supervisor for GD_N, then S_{NTRi} is a non-blocking supervisor for GD_{NTiRi}. □

If the non-blocking condition holds for all i = 1, 2, ..., p, the fault recovery supervisor is a non-blocking supervisor.

3.2.6 Supervisor Admissibility

As usual, suppose S_N and S_T are admissible for GD_N and GD_{NT}, respectively. It is important that the normal supervisor (S_N) be an admissible controller for GD_{NT} as well. In other words, care must be taken that S_N not disable any uncontrollable event in both normal and transient modes.

To verify the admissibility of S_N with respect to GD_{NT}, the **condat** procedure may be used.

$$\mathbf{STDAT = condat(GD_{NT}\ ,\ S_{NT})}$$

STDAT displays a list of supervisor (S_{NT}) states in which the disabling of events may occur. If any uncontrollable event is disabled by S_{NT}, then S_{NT} will not be an admissible supervisor for GD_{NT}. Note that by assumption, S_T is an admissible supervisor for GD_{NT} and therefore does not disable uncontrollable events. Thus, disablement of uncontrollable events, if any, is done by S_N.

In order to prevent S_N from disabling uncontrollable events in the transient mode, the following procedure may be performed.

Let $\Sigma_{UC} \subseteq \Sigma_P$ denote the uncontrollable events.

1. Add a dump state (X_d) to S_N and mark it.

2. For any state of S_N, q, and $\sigma \in \Sigma_{UC}$ such that σ is not defined at q in S_N, add a transition (q , σ , X_d) from q to the dump state.

3. Attach a transition (X_d, α , X_d) for any event $\alpha \in \Sigma_P \cup \Sigma_f \cup \Sigma_d$ at this state.

Call the modified supervisor S_{NM}. S_{NM} never disables an uncontrollable event. The transitions added in step 2 may only occur during the transient mode, after which S_{NM} enters its dump state and is effectively taken out of the control loop. Therefore, $L(S_N/GD_N) = L(S_{NM}/GD_N)$ and $L_m(S_N/GD_N) = L_m(S_{NM}/GD_N)$, i.e., S_N and S_{NM} have the same effect on the plant during the normal mode.

From now on, we assume that whenever S_N is not admissible for GD_{NT}, it is replaced by S_{NM}. In those cases, for simplicity, we drop M and denote the resulting supervisor by S_N.

In the case of multiple failures (p>1), S_{Ri} is designed based on GD_{NTiRi}. Therefore, we have to verify that S_{Ri} is admissible for GD_{NT} and if the condition is not satisfied, the above mentioned procedure must be applied to S_{Ri} to render it admissible.

Example 3.1 (Cont'd): We verify that IS_{NT} and IS_R are admissible supervisors, and the resulting modular switching supervisor is non-blocking

1. Admissibility property

$$ISNTGDNTDAT = \textbf{condat}(IGD_{NT}, IS_{NT})$$

$$ISRGDNTDAT = \textbf{condat}(IGD_{NT}, IS_R)$$

The result of ISNTGDNTDAT and ISRGDNTDAT are shown in Fig. 3.29. It can be seen that in both cases, every disabled event is controllable. Thus, both supervisors are admissible for IGD_{NT}.

```
ISNTGDNTDAT

Control data are displayed as a list of supervisor states
where disabling occurs, together with the events that must
be disabled or forced there.

control data:

       0:   5                          1:   1    5
       2:   1                          5:   1
       7:   5                          8:   1
      11:   1


ISRGDNTDAT

Control data are displayed as a list of supervisor states
where disabling occurs, together with the events that must
be disabled or forced there.

control data:

       0:   5                          1:   5
       5:   5                         24:   5
      52:   1                        122:   1
     137:   1
```

Figure 3.29: Example 3.1. Results of **condat** procedure

2. Non-blocking

$$\textbf{nonconflict}(\ IGD_N\ ,\ \textbf{meet}(IS_{NT}\ ,\ IS_R)) = \text{true}\ ?$$

The answer to the above question is "yes", therefore, the conjunction of IS_{NT} and IS_R is a non-blocking supervisor for IGD_N.

so **meet**(IS_{NT} , IS_R) is a non-blocking supervisor.

We also use the following **nonconflict** procedure for IGD_{NTR} and IS_R, and find that IS_R is a non-blocking supervisor for IGD_{NTR}.

$$\textbf{nonconflict}(\ IGD_{NTR}\ ,\ IS_R) = \text{true}\ ?$$

Finally, for the transfer line, since **meet**(IS_{NT} , IS_R) is a non-blocking supervisor for IGD_N, and IS_R is a non-blocking supervisor for IGD_{NTR}, by Theorem 3.1, we conclude that IS_{NTR} is a non-blocking supervisor for IGD_{NTR}.

In conclusion, the modular supervisor is a non-blocking supervisor.

3.2.7 Example 3.2: Tank–Valve System

In this example, we demonstrate the application of the framework developed in this thesis to the design of recovery controllers for a tank-valve system. The example involves two faults and illustrates recovery in systems having redundant components. The system (Fig. 3.30) consists of a tank (Tank), two control valves (V1,V2), and one mechanical valve

(V3). Valve V2 is redundant. When valve V1 is in the normal (non-faulty) condition, valve V2 is in standby mode and not in service.

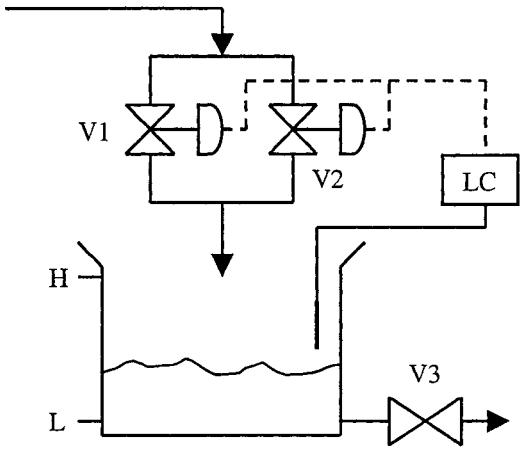


Figure 3.30: Example 3.2. Tank-Valve System

We assume that the liquid level in the tank is initially in the low range, below level low point (L). Valve V3 is open to supply liquid to another process. If the liquid level goes below the point L, then valve V1 is opened and the liquid fills the tank until it reaches point H. At this point, valve V1 is closed, the level drops to below L and the cycle is complete. Following this, another cycle starts.

It is assumed that the system is equipped with a level sensor that can read one of the two possible values: level low and level high. Furthermore, it is assumed that valve V2 works properly and never fails. Furthermore, valve V1 has two failure modes, a ``*stuck-open*`` failure mode and a ``*stuck-closed*`` failure mode. Valve V1 can get stuck-closed only when it is closed, and it can get stuck-open only when it is open.

In this system, the controller is responsible for opening and closing the valve when the liquid level goes below L or above H.

The automata for valves V1 and V2, tank, and controller (Ctrl) are depicted in Fig. 3.31.

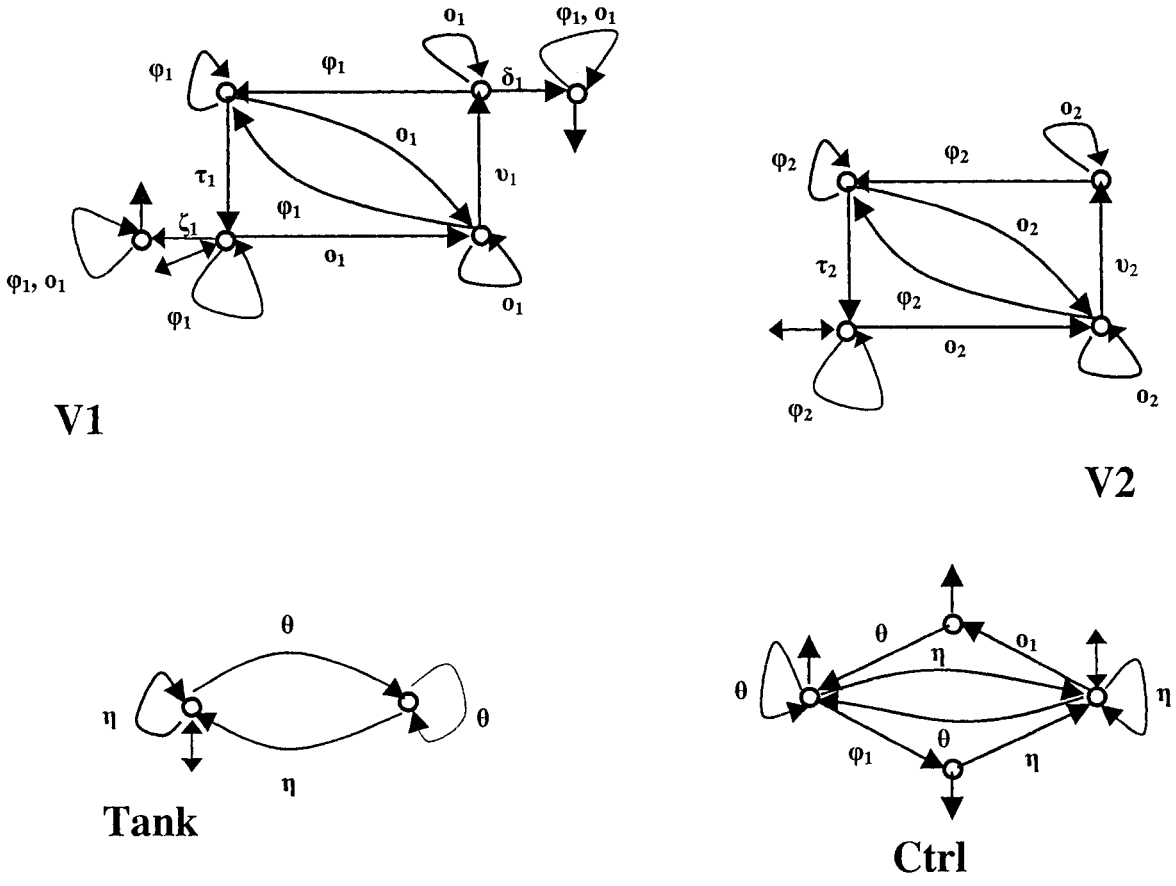


Figure 3.31: Example 3.2. DES models of valves V1 and V2, tank, and controller

To find a DES model for the interlock between tank and valves, first the synchronous product of the valves is computed. Then this product is modified by adding selfloops of events level low (η) and level high (θ) to the states corresponding to valves open or stuck-open, and valves closed or stuck-closed, respectively. The DES structure of the interlock between valves and tank is displayed in the next subsections.

Another element in this plant is the *emergency shut down system.* It is assumed that if valve V1 gets STUCK-OPEN then the system must be shut down. This could be simply closing an emergency valve upstream. The DES model of the emergency shut down system (ESM) is shown in Fig. 3.32.

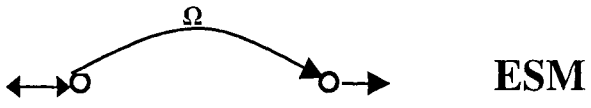


Figure 3.32: Example 3.2. DES model of the emergency shut down system

Table 3.3: Example 3.2. Events and their description

Event	Description	C/UC
o_1	VALVE_V1_OPEN	C
υ_1	VALVE_V1_OPEN	UC
φ_1	VALVE_V1_CLOSE	C
τ_1	VALVE_V1_CLOSED	UC
δ_1	VALVE_V1_STUCK_OPEN	UC
ζ_1	VALVE_V1_STUCK_CLOSED	UC
χ_1	VALVE_V1_STUCK-OPEN_DETECTED	UC
ω_1	VALVE_V1_STUCK-CLOSED_DETECTED	UC
o_2	VALVE_V2_OPEN	C
υ_2	VALVE_V2_OPEN	UC
φ_2	VALVE_V2_CLOSE	C
τ_2	VALVE_V2_CLOSED	UC
η	TANK-LEVEL_LOW	UC
θ	TANK-LEVEL_HIGH	UC
Ω	SHUT_DOWN	C

A complete list of events is given in Table 3.3. In this table C and UC denote controllable and uncontrollable, respectively. Moreover, in this example, the list of recovery events is $\Sigma_r = \{\upsilon_2, o_2, \varphi_2, \tau_2, \Omega\}$.

The objective of the above problem is to compute recovery controllers in order to reconfigure the system in case of V1 faults. The design procedure is as follows.

Normal Mode

Let NV1 and InterlockN (shown in Fig. 3.33) respectively be the normal model of valve V1 and its interlock with the tank. IIG_N, the normal model of the plant, is constructed as follows.

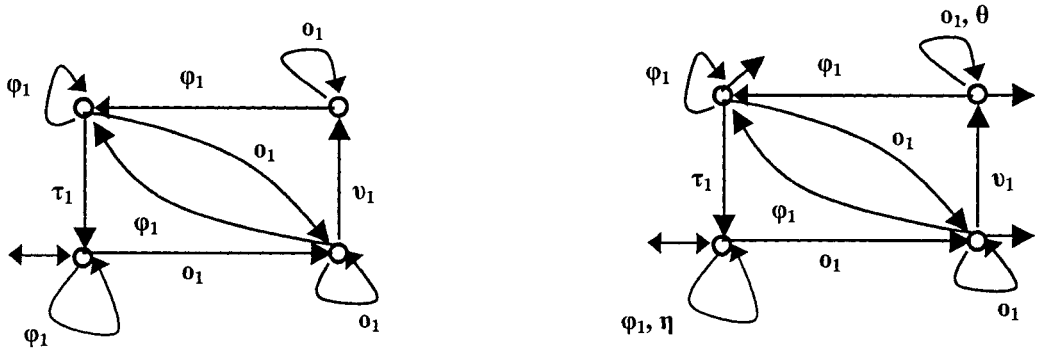


Figure 3.33: Example 3.2. Normal models of valve V1 and its interlock with the tank

TCIN = sync(Tank , Ctrl , InterlockN)

$IIGD_N$ = sync(NV1 , TCIN)

The normal specification, which is formalized as IIE_N in Fig. 3.34, is the following.

i. V1 should be active.

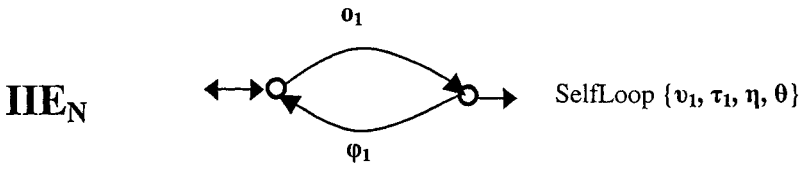


Figure 3.34: Example 3.2. Normal specification

Finally, the normal supervisor displayed in Fig. 3.35 is IIS_N where

$$IIS_N = supcon(IIGD_N , IIE_N).$$

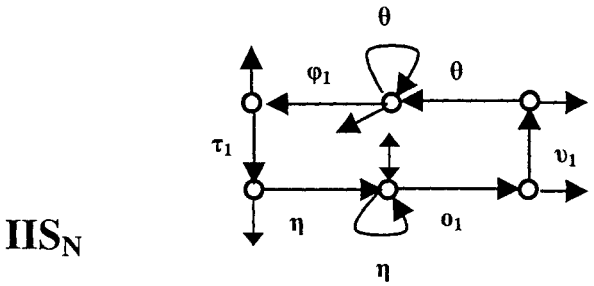


Figure 3.35. Example 3.2: Normal supervisor

Normal-Transient Subsystem

The following models are considered for valve V1 and its interlock with the tank for building $IIGD_{NT}$.

$$TCIT = sync(Tank , Ctrl , InterlockNT)$$

$$IIG_{NT} = sync(TV1 , TCIT)$$

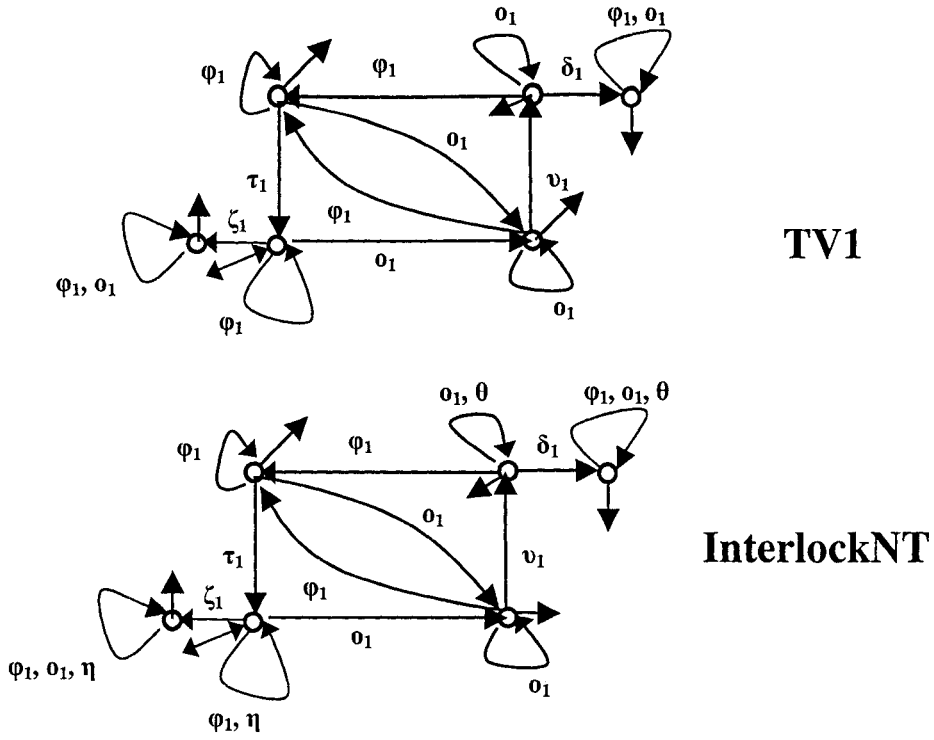


Figure 3.36: Example 3.2. Models for valve V1 and its interlock with the tank

We assume that the detection delay of the diagnoser in this example is at most one event.

The DES models of these diagnosers are depicted in Fig. 3.37.

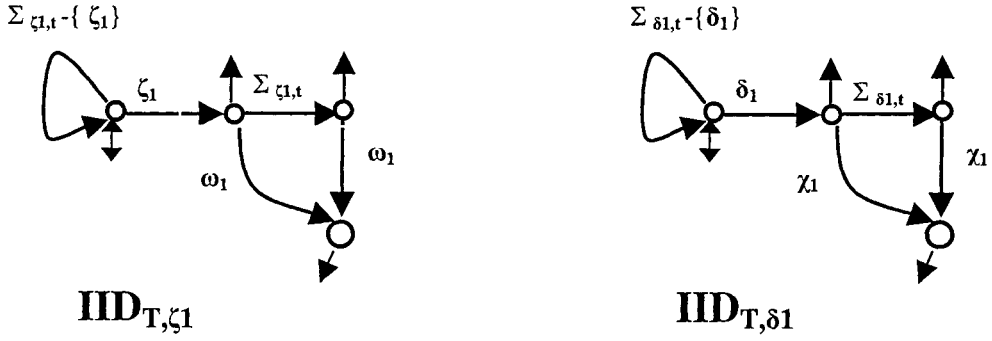


Figure 3.37: Example 3.2. Modular DDM in transient mode

Here

$$\Sigma_{\zeta 1,t} = \Sigma_p \cup \Sigma_f \cup \{\chi_1\} = \{\upsilon_1, o_1, \varphi_1, \tau_1, \zeta_1, \delta_1, \chi_1, \eta, \theta\}$$

$$\Sigma_{\delta 1,t} = \Sigma_p \cup \Sigma_f \cup \{\omega_1\} = \{\upsilon_1, o_1, \varphi_1, \tau_1, \zeta_1, \delta_1, \omega_1, \eta, \theta\}$$

The diagnoser $\mathbf{IID_{NT}}$ will be **sync(** $\mathbf{IID_{T,\zeta 1}}$ **,** $\mathbf{IID_{T,\delta 1}}$ **)**.

The combined model of the plant and diagnoser for the normal and transient modes is:

$$\mathbf{IIGD_{NT} = sync(\ IIG_{NT}\ ,\ IID_{NT}\ )}$$

In the transient mode, we assume the same specification as in the normal mode. The automaton model of this specification in the transient mode is $\mathrm{IIE_N}$ selflooped with the fault and detection events.

$$\mathbf{IIE_T = selfloop(IIE_N\ ,\ \{\zeta_1, \delta_1, \omega_1, \chi_1\})}$$

The procedure for computing the transient supervisor is as follows:

1. **IITN = supnorm(** $\mathbf{IIE_T}$ **,** $\mathbf{IIGD_{NT}}$ **, NULL)** , **NULL=** $\{\zeta_1, \delta_1\}$
2. **IITNO = project(IITN , NULL)**
3. **IITGO = project(** $\mathbf{IIGD_{NT}}$ **, NULL)**
4. **IITKO = supcon(IITGO , IITNO)**
5. **IITKODAT = condat(IITGO , IITKO)**
6. $\mathbf{IIS_T}$ **= selfloop(IITKO , NULL)**
7. **IITK = meet(**$\mathbf{IIGD_{NT}}$ **,** $\mathbf{IIS_T}$**)**
8. **IITK** nonempty **?**

$\mathrm{IIS_{NT}}$, the synchronous product of the normal and transient supervisors, is:

$$\mathbf{IIS_{NT} = sync(IIS_N\ ,\ IIS_T)}$$

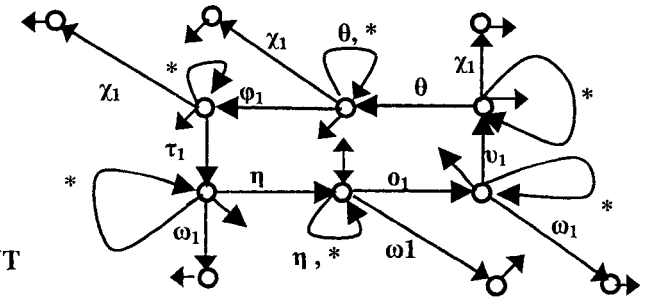


Figure 3.38: Example 3.2. Normal-transient supervisor

Recovery Mode

Since we assume that there are two faults in this system, two different recovery supervisors must be computed, one for recovery from the stuck-closed fault, and the other for recovery from the stuck-open fault.

For the case of "VALVE_V1_STUCK_CLOSED" fault, in order to find the plant's recovery model, RV1 and V2 are considered to be the models for valves V1 and V2, respectively. To model the interlock (**InterlockNT1R1**) among the valves (V1 and V2) and the tank, first, the synchronous product of RV1 and V2 is computed. Next the selfloops of events TANK-LEVEL-LOW and TANK-LEVEL-HIGH are added to states corresponding to valve open, and valve closed or stuck-closed, respectively.

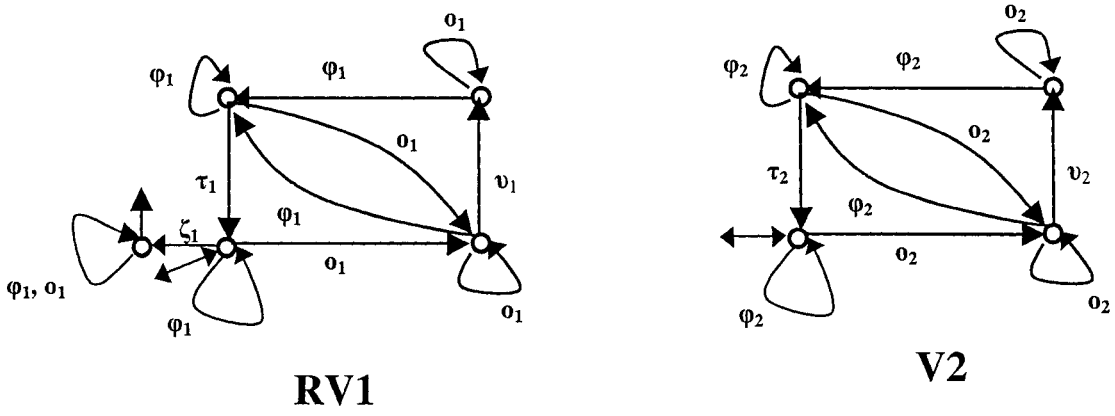


Figure 3.39: Example 3.2. Models for valve V1 and V2 for the recovery mode

In addition, in this case, we consider one controller for each valve (V1 and V2). These models are depicted in Fig. 3.40.

Finally, we construct the following automata:

$$\mathbf{CtrlR = sync(CtrlR1\ ,\ CtrlR2)}$$

$$\mathbf{TCIR1 = sync(Tank\ ,\ CtrlR\ ,\ InterlockNT1R1)}$$

$$\mathbf{IIG_{NT1R1} = sync(RV1\ ,\ V2\ ,\ TCIR1)}$$

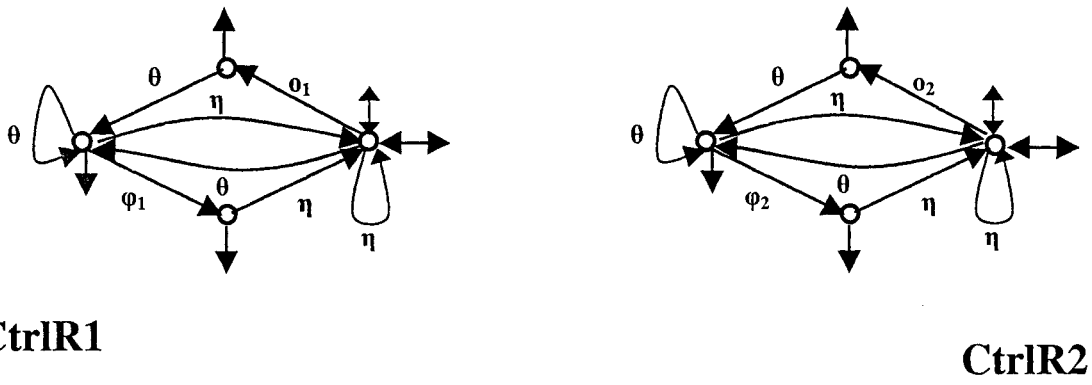


Figure 3.40: Example 3.2. Controllers for V1 and V2 in recovery mode

For the case "VALVE_V1_STUCK_OPEN " fault, in order to find the plant's recovery model, we model V1 by RV2 (shown in Fig 3.41). Also, to model the interlock (**InterlockNT2R2**) among valve V1, the emergency shutdown system, and the tank, first, the synchronous product of V1 and ESM is computed. Next the selfloops of events TANK-LEVEL-LOW and TANK-LEVEL-HIGH are added to states corresponding to valve open, stuck-open, and emergency-shutdown-on, valve closed, respectively.

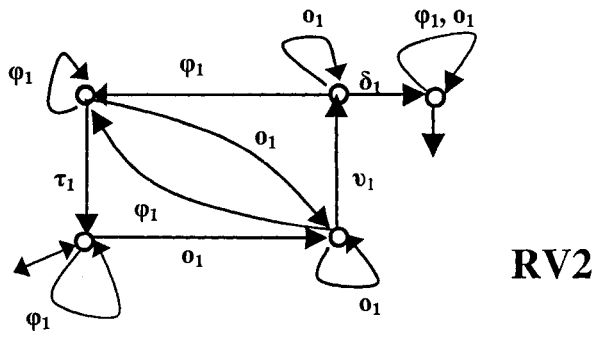


Figure 3.41: Example 3.2. Model for valve V1 in the recovery mode

$$\mathbf{TCIR = sync(Tank\ ,\ Ctrl\ ,\ InterlockNT2R2\ ,\ ESM)}$$

$$\mathbf{IIG_{NT2R2} = sync(RV2\ ,\ TCIR)}$$

In the recovery mode, the models of the diagnosers is like that in the transient mode, except that we add the selfloops at detection states and modify the events sets by adding recovery events to $\Sigma_{\zeta 1,t}$ and $\Sigma_{\delta 1,t}$.

$$\Sigma_{\zeta 1,r} = \Sigma_p \cup \Sigma_f \cup \{\chi_1\} \cup \Sigma_r = \{\upsilon_1, o_1, \varphi_1, \tau_1, \zeta_1, \delta_1, \chi_1, \eta, \theta, \upsilon_2, o_2, \varphi_2, \tau_2, \Omega\}$$

$$\Sigma_{\delta 1,r} = \Sigma_p \cup \Sigma_f \cup \{\omega_1\} \cup \Sigma_r = \{\upsilon_1, o_1, \varphi_1, \tau_1, \zeta_1, \delta_1, \omega_1, \eta, \theta, \upsilon_2, o_2, \varphi_2, \tau_2, \Omega\}$$

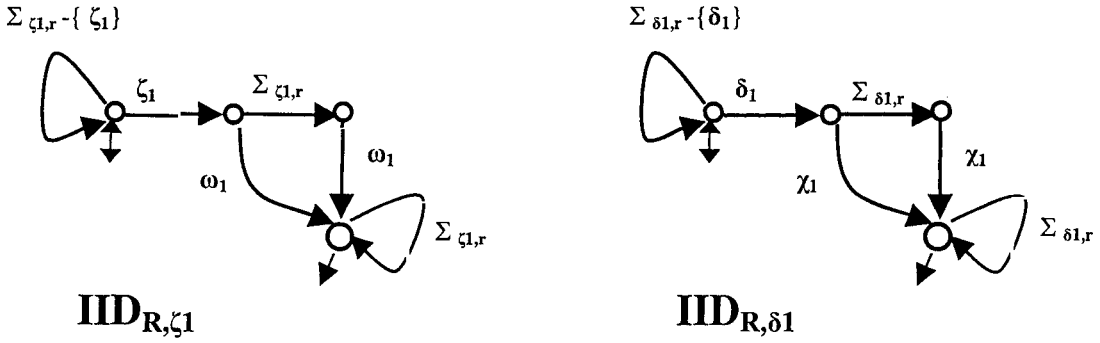


Figure 3.42: Example 3.2. Modular DDM in recovery mode

Finally, in the recovery mode, the combined model of the plant and diagnosers (the system to be controlled) corresponding to faults ζ_1 and δ_1 are as follows.

$$\mathbf{IIGD_{NT1R1} = sync(\ IIG_{NT1R1}\ ,\ IID_{R,\zeta 1})}$$

$$\mathbf{IIGD_{NT2R2} = sync(\ IIG_{NT2R2}\ ,\ IID_{R,\delta 1})}$$

The specification considered for recovering the system from the fault VALVE_V1_STUCK_CLOSED is

• If valve V1 gets stuck-closed then the recovery controller must send open and close commands to valve V2, which replaces V1.

This specification is formalized in Fig. 3.43.

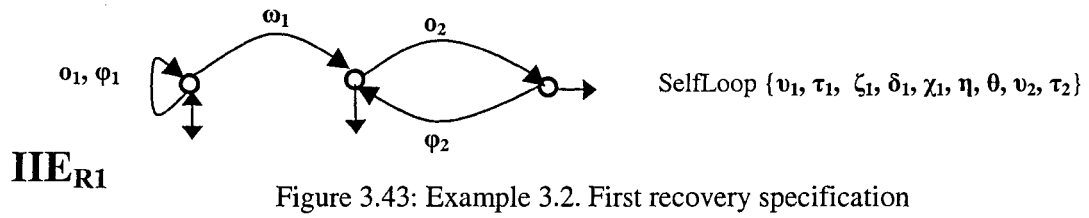


Figure 3.43: Example 3.2. First recovery specification

Another specification used in recovery mode is the one considered for recovery from the fault VALVE_V1_STUCK_OPEN. This specification (Fig. 3.44) states that

• If valve V1 gets stuck-open then the system must be shut down. In other words, the controller must stop sending commands to valve V1 (no command to valve V2).

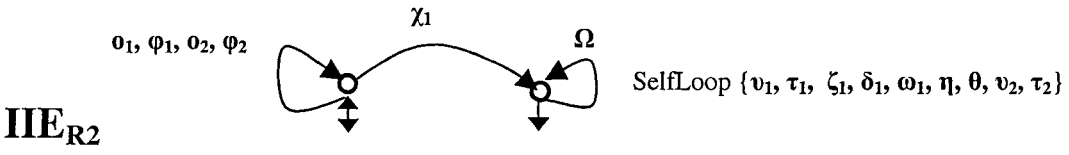


Figure 3.44: Example 3.2. Second recovery specification

The procedure of computing the recovery supervisor corresponding to fault ζ_1 is as follows:

1. **IIRN1 = supnorm(IIE_{R1} , $IIGD_{NT1R1}$, NULL)** , **NULL= { ζ_1 }**
2. **IIRNO1 = project(IIRN1 , NULL)**
3. **IIRGO1 = project($IIGD_{NT1R1}$, NULL)**

4. **IIRKO1 = supcon(IIRGO1 , IIRNO1)**
5. **IIRKODAT1 = condat(IIRGO1 , IIRKO1)**
6. **IIS_{R1} = selfloop(IIRKO1 , { ζ_1 , δ_1})**
7. **nonconflict($IIGD_{NT1R1}$, IIS_{R1})** = true ?
8. **IIRK1 = meet($IIGD_{NT1R1}$, IIS_{R1})**
10. **IIRK1** nonempty ?

The same procedure can be used to compute the second recovery supervisor corresponding to fault δ_1. This procedure uses IIE_{R2} (the second recovery specification) instead of IIE_{R1}.

1. **IIRN2 = supnorm(IIE_{R2} , $IIGD_{NT2R2}$, NULL)** , **NULL= { δ_1}**
2. **IIRNO2 = project(IIRN2 , NULL)**
3. **IIRGO2 = project($IIGD_{NT2R2}$, NULL)**
4. **IIRKO2 = supcon(IIRGO2 , IIRNO2)**
5. **IIRKODAT2 = condat(IIRGO2 , IIRKO2)**
6. **IIS_{R2} = selfloop(IIRKO2 , { ζ_1 , δ_1})**
7. **nonconflict($IIGD_{NT2R2}$, IIS_{R2})** = true ?
8. **IIRK2 = meet($IIGD_{NT2R2}$, IIS_{R2})**
9. **IIRK2** nonempty ?

Both of the above supervisors, IIS_{R1} and IIS_{R2}, are admissible and non-blocking with respect to their plants, $IIGD_{NT1R1}$ and $IIGD_{NT2R2}$.

Decoupling Condition

In this example, the decoupling condition is verified for the two recovery supervisors. Because the verification procedure is the same as the transfer line example, for brevity, we only include the results of the verifications.

$$\mathbf{L}\,(\,\mathbf{IIS_{NT}}/\,\mathbf{IIGD_{NT}})\ \subseteq\ \mathbf{L}\,(\,\mathbf{IIS_{R1}}/\,\mathbf{IIGD_{NT}})\qquad \equiv \text{true}$$

$$\mathbf{L}\,(\,\mathbf{IIS_{NT}}/\,\mathbf{IIGD_{NT}})\ \subseteq\ \mathbf{L}\,(\,\mathbf{IIS_{R2}}/\,\mathbf{IIGD_{NT}}))\qquad \equiv \text{true}$$

This means that the decoupling condition is satisfied for both recovery supervisors.

Supervisor Verification

The aim of this section is to verify whether or not the produced normal, transient and recovery supervisors (IIS_{NT}, IIS_{R1}, and IIS_{R2}) are admissible and that the system under supervision is non-blocking. We begin with admissibility property.

1. Admissibility property

Here, we need only to perform the following computations.

$$IISNTGDNTDAT = \mathbf{condat}(\,IIGD_{NT}\,,\,IIS_{NT})$$

$$IISR1GDNTDAT = \mathbf{condat}(\,IIGD_{NT}\,,\,IIS_{R1})$$

$$IISR2GDNTDAT = \mathbf{condat}(\,IIGD_{NT}\,,\,IIS_{R2})$$

Since IISNTGDNTDAT, IISR1GDNTDAT, and IISR2GDNTDAT do not contain any state in which the uncontrollable events are prevented from occurrence by the supervisors, IIS_{NT}, IIS_{R1}, and IIS_{R2} are admissible with respect to $IIGD_{NT}$.

2. Non-blocking property

In order to check this property, we use the **nonconflict** procedure and find that the answer to the following question is "yes". Thus, IIS_{NTR} is a non-blocking supervisor with respect to $IIGD_N$.

$$\mathbf{nonconflict}(\,IIGD_N\,,\,\mathbf{meet}(IIS_{NT}\,,\,IIS_{R1},\,IIS_{R2})) = \text{true}\ ?$$

The recovery supervisors are $\mathbf{IIS_{R1}}$ and $\mathbf{IIS_{R2}}$. Let

$$IIS_{NTR1} = \mathbf{meet}(\text{meet}(II\,\tilde{S}_{NT}, II\,\tilde{S}_{R2}), IIS_{R1})$$

$$IIS_{NTR2} = \mathbf{meet}(\text{meet}(II\,\tilde{S}_{NT}, II\,\tilde{S}_{R1}), IIS_{R2})$$

Since

i. $\mathbf{meet}(\mathbf{IIS_{NT}}, \mathbf{IIS_{R1}}, \mathbf{IIS_{R2}})$ is a non-blocking supervisor for $\mathbf{IIGD_N}$

ii. $\mathbf{IIS_{R1}}$ and $\mathbf{IIS_{R2}}$ are non-blocking supervisors for $\mathbf{IIGD_{NT1R1}}$ and $\mathbf{IIGD_{NT2R2}}$, respectively

then $\mathbf{IIS_{NTR1}}$ and $\mathbf{IIS_{NTR2}}$ are non-blocking with respect to $\mathbf{IIGD_{NT1R1}}$, $\mathbf{IIGD_{NT2R2}}$, respectively. Then by Theorem 3.2 the modular switching supervisor for the Tank-Valve system is a non-blocking supervisor.

In this example, after recovery, Valve 2 receives the open and close commands in order to fill up the tank. It does the same job as valve 1 was doing before. In this case, the system can be considered to be in the normal mode again. Let us assume valve 2 may fail. Then there could be two consecutive faults in tank-valve system. In this case, we could add another normal, transient, and recovery modes to the system model. Fig.3.45 shows the relations among the modes.

$\mathbf{GD_N^{(1)}}$ and $\mathbf{GD_T^{(1)}}$ respectively indicate the system in the normal, transient, and recovery modes corresponding to the first fault. $\mathbf{GD_T^{(2)}}$, and $\mathbf{GD_R^{(2)}}$ respectively denote the system in the transient, and recovery modes corresponding to the second fault. $\mathbf{GD_R^{(1)}}$ describes

the condition of the system after recovering from the first fault and before the occurrence of the second fault. Therefore, we may refer to it as $\mathbf{GD_N^{(2)}}$, as well.

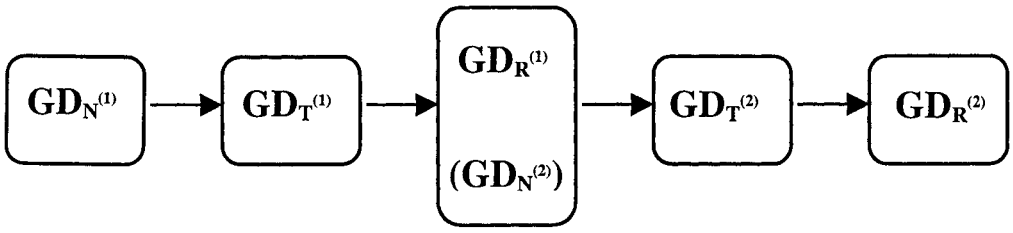


Figure 3.45: Example 3.2. Tank-Valve system for two consecutive faults

In order to model the system under supervision and design controllers, the same procedure can be repeated. This is further explored in the following.

Remark 3.3: Return to Normal Mode

Let us consider a recovery problem in which after recovery the plant returns to its normal mode. We assume that after the occurrence of a recovery event (F_R), the plant goes to one of its normal states. This second normal mode is designated by $GD_{N'}$.

The timing diagram in Fig. 3.46 displays the recovery procedure. For simplicity, we assume a single failure system (p=1). $S_{N'}$ is a controller that is synthesized in order to restrict the behavior of $GD_{N'}$; that is, to confine it to a modified version of normal specifications (to be explained later). This controller is another modular supervisor that is added to the control system, and it performs a role similar to what the recovery supervisor does during the normal, transient, and recovery modes. In other words, $S_{N'}$ follows the sequence of events generated by the plant under control of other supervisors. In general, it may restrict the plant behavior to make sure that the normal specifications can be met

after recovery. When the recovery event happens in the system, then $S_{N'}$ will take sole control of the plant.

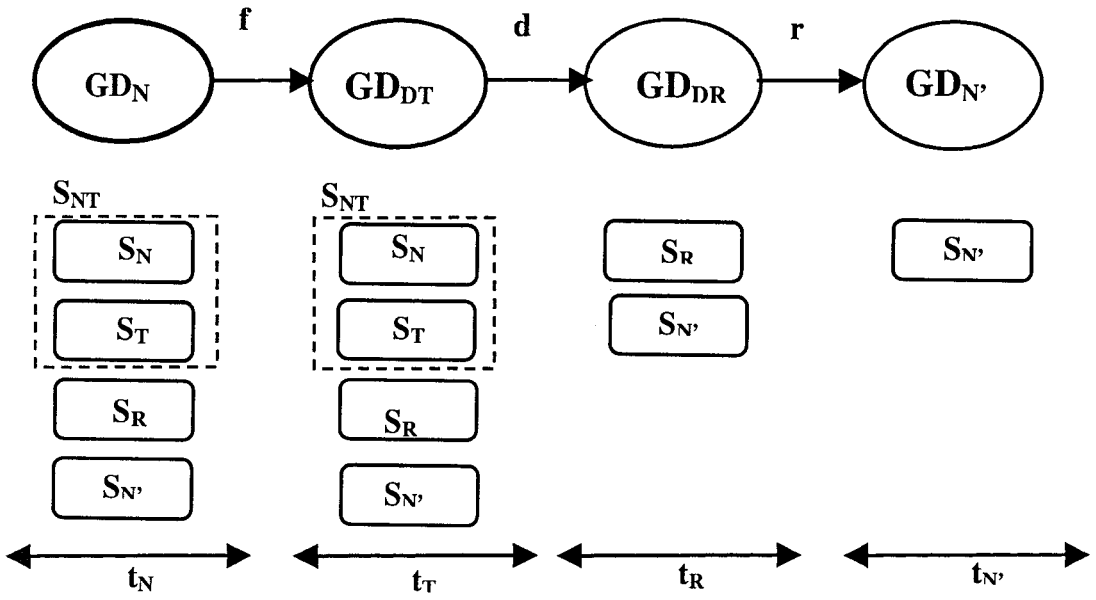


Figure 3.46. Timing diagram of the developed recovery procedure

Fig. 3.47 shows how to construct $GD_{N'}$. It is composed of GD and GD_N connected with recovery events. The recovery transitions to normal states depend on the type of the fault as well as the plant behavior. For example, after recovery, a component may return to its initial state while another component may resume its operation which might have been interrupted due to fault.

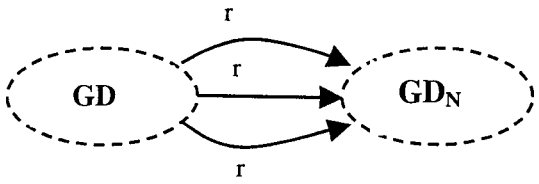


Figure 3.47: $GD_{N'}$

Chapter 4

Fault Recovery in Timed Discrete Event Systems

4.1 Overview

In this chapter, we extend our fault recovery approach to timed discrete event systems. TDES models can be used in many real-time computer systems and control engineering problems where a methodology to ensure the correctness of the operation of real-time systems is important. These modules can be composed together to produce multipart systems. Therefore, they are useful in modeling complex systems. An important feature of our framework, namely the delay model for the diagnoser, becomes more useful and significant in problems involving timing.

4.2 Recovery Framework

Our recovery framework for TDES basically uses the same methodology presented in the previous chapter with the exception that the formalism of TDES models is used to analyze the plant and control systems. In this study, the clock tick is considered as an extra event. The definitions of the normal, transient, and recovery modes are the same as those in chapter 3. For the computation of the supervisors, the theory of supervisory control of timed discrete–event systems in [12] and its extension to supervisory control of TDES under partial observation presented in [15] are used.

Our discussion in this chapter will be limited to single failure case; the result can be generalized to the case of multiple failures as in chapter 3.

4.2.1 Plant Model

The plant under supervision is assumed to be a timed discrete event system (TDES). Specifically, the plant under supervision is defined to be a 5-tuple

$$G_\tau = (Q_\tau , \Sigma_\tau , \delta_\tau , q_\circ, Q_m),$$

where Q_τ , Σ_τ , $q_\circ$, and Q_m are the finite state set, the set of events, the initial state, and the set of marked states, respectively. $\delta_\tau : Q_\tau \times \Sigma_\tau \rightarrow Q_\tau$ is the state transition partial function.

In this study, Σ_τ is partitioned into the following four subsets: $\Sigma_\tau = \{\text{tick}\} \cup \Sigma_p \cup \Sigma_r \cup \Sigma_f$. Σ_f and Σ_r are the sets of fault and recovery events, respectively. The set of the remaining non-tick events is denoted by Σ_p.

G_τ describes the system behavior in both normal and faulty conditions. It is assumed that G_τ contains P permanent failure modes F_1, ..., F_P. We define $K=\{N, F_1, ..., F_P\}$ as the condition set. The state set of G_τ can be partitioned according to the condition of the system: $Q = Q_N \mathbin{\mathring{\cup}} Q_{F1} \mathbin{\mathring{\cup}} .... \mathbin{\mathring{\cup}} Q_{FP}$.

We assume that the timed transition graph (TTG) of TDES model of G_τ is obtained from an activity transition graph (ATG), G_{act}, with

$$G_{act} = (A , \Sigma_{act} , \delta_{act} , a_\circ, A_m)$$

where A is the activity set, Σ_{act} is a finite set of events, $\delta_{act} : A \times \Sigma_{act} \rightarrow A$ is the activity transition partial function, $a_\circ$ is the initial activity, and A_m is the set of marked activities.

It should be mentioned that we assume that the tick event is observable, and it cannot be disabled by supervisors, but it can be preempted by forcible events. We should also mention that from now on, τ is used to denote the tick.

4.2.2 Model for Fault Diagnosis System

In this section, the model presented in the previous chapter for fault diagnosis system is extended to the timed discrete event systems. Similar to the untimed case, we define a diagnostic delay model (DDM) for the diagnosis system. For TDES, instead of using events to represent the delay of the diagnoser, we use the time bounds in ticks to measure

and represent diagnosis delay. These time bounds are simply the lower and upper time bounds of the detection events.

The ATG of DDM is a 5-tuple

$$D_{act} = (A_D, \Sigma_{Dact}, \delta_{Dact}, a_{D\circ}, A_{D,m}),$$

where A_D is the activity set, Σ_{Dact} is a finite set of events, $\delta_{Dact} : A_D \times \Sigma_{Dact} \rightarrow A_D$ is the activity transition partial function, $a_{D\circ}$ is the initial activity, and $A_{D,m}$ is the set of marked activities. $\Sigma_{Dact} = \Sigma_{Gact} \cup \Sigma_d$ consists of the plant and detection events. Fig. 4.1 displays a general DDM.

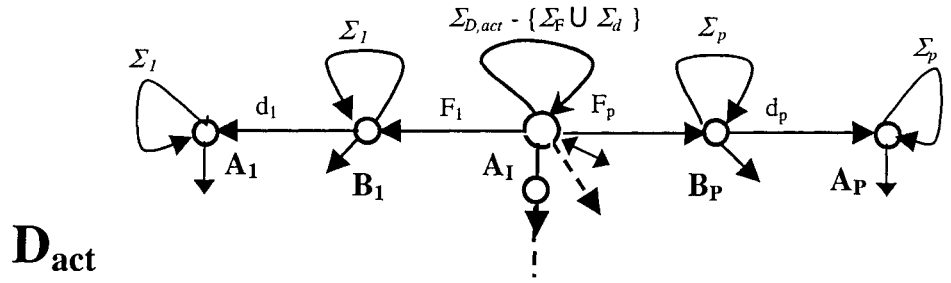


Figure 4.1: A general DDM in TDES

where $\Sigma_1 = \Sigma_{Dact} - \{d_1\}$, ..., $\Sigma_p = \Sigma_{Dact} - \{d_p\}$; $F_1, F_2, \ldots, F_p$ are unobservable fault events and $d_1, d_2, \ldots, d_p$ are the corresponding detection events, with time bounds (l_{d1}, u_{d1}) , (l_{d2}, u_{d2}), ..., (l_{dp}, u_{dp}). The upper bounds u_{di} are assumed finite ($u_{di} < \infty$). In other words, the detection events are prospective. Otherwise, the failures may not be diagnosed at all.

The DDM model functions as follows. Initially, when the plant is in its normal states, the DDM is in A_I. If a fault, say, F_p happens in the system, then the diagnoser goes to its next state, Bp. At this state, either the diagnoser detects the fault and generates the detection event d_p, or the plant continues to generate other events from its set of events. Eventually before u_{dp}+1 clock ticks, the diagnoser detects the fault and enables the detection event and goes to the next state Ap.

The modular models of DDM for two faults F_1 and F_2 are shown in Fig. 4.2. These models can also be developed for more faults similarly.

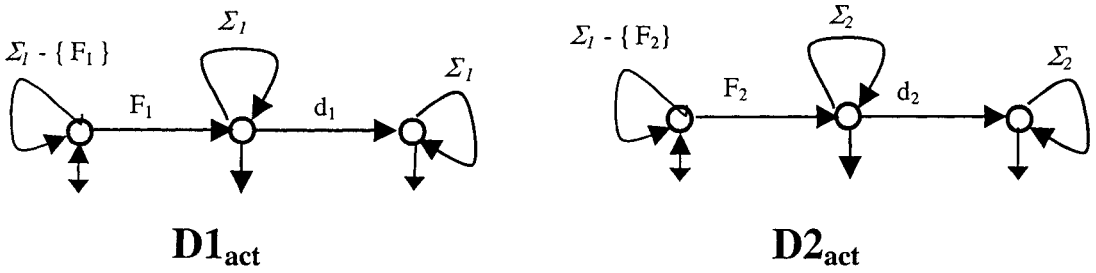


Figure 4.2: Modular DDM in TDES

The equivalent DDM for the system in Fig. 4.2 is

$$\mathbf{D_{act} = comp(\ D1_{act}\ ,\ D2_{act})}$$

The TTG of the TDES can be obtained from

$$\mathbf{D\ = Timed\text{-}Graph(\ D_{act})}$$

4.2.3 The System to Be Controlled

The combined model of the plant and diagnoser (the system to be controlled) is the composition of the plant (G) and the diagnosis system (D). It should be mentioned that for the composition of TDES, we follow the notation described in chapter 2.

$$\mathbf{GD_{act} = comp(\ G_{act}\ ,\ D_{act})}$$

and

$$\mathbf{GD = Timed\text{-}Graph(\ GD_{act})}$$

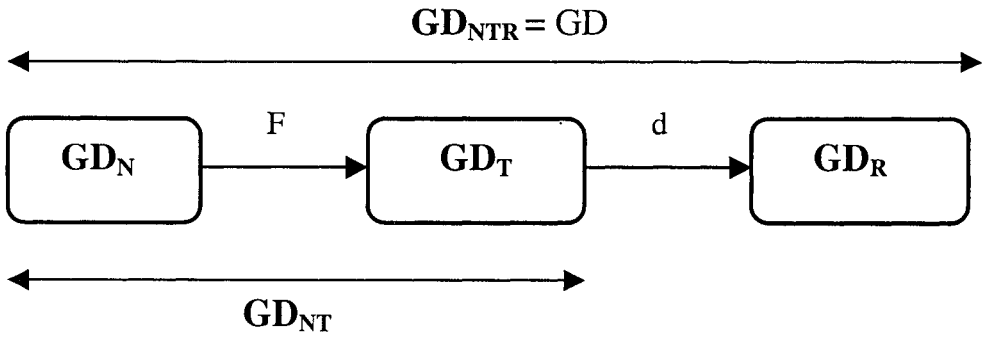


Figure 4.3: GD has three modes (assuming single failure, p=1)

Let $\Sigma = \Sigma_G \cup \Sigma_D = \Sigma_G \cup \Sigma_d$ denote the event set of GD. GD has three modes: normal, transient, and recovery (Fig. 4.3). G_{DN} is the plant and the diagnoser in the normal mode. GD_{NT} describes the plant in normal and transient modes. GD_{NTR} denotes the plant in normal, transient, and recovery modes.

4.2.4 Example 4.1: Small factory

A small factory includes two *machines* and a *buffer* with one slot. The arrangement of the small factory is shown below:

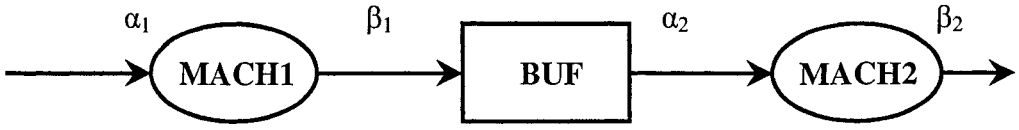


Figure 4.4: Example 4.1. Small factory

MACH1, **BUF**, **MACH2** illustrate *machine* 1, *buffer,* and *machine* 2, respectively. The ATG models of the machines are shown in Fig. 4.5.

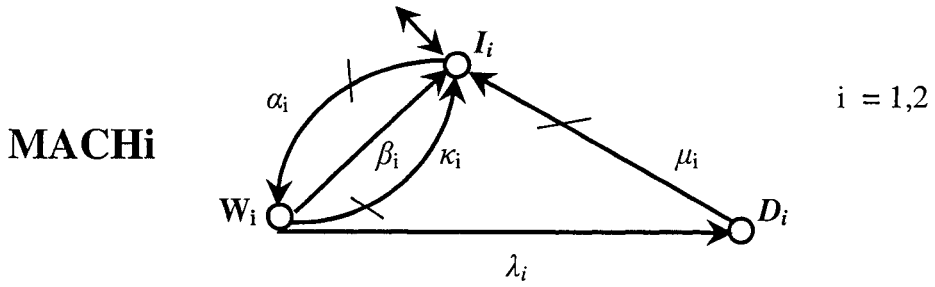


Figure 4.5: Example 4.1. DES model of machines 1 and 2

The operation of the small factory is as follows. Initially the buffer is empty. With the event α_1, **MACH1** takes a workpiece from an infinite input bin and enters $\mathbf{W_1}$. Subsequently **MACH1** either breaks down and enters $\mathbf{D_1}$ (event λ_1), or successfully completes its work cycle, deposits the workpiece in the buffer, and returns to $\mathbf{I_1}$ (event β_1). **MACH2** operates similarly, but takes its workpiece from the buffer and deposits it when finished, in an infinite output bin. Upon repair, a machine returns to its initial state. Moreover, when a machine is in its work positions (state $\mathbf{W_i}$), it can be reset (event κ_i) to its initial position (state $\mathbf{I_i}$).

The informal specifications for admissible operation are:

1. The buffer must not overflow or underflow.

2. If both machines break down, then **MACH2** must be repaired before **MACH1.**

3. A machine can be reset from work state to ideal (event κ_i) if the other machine has broken down.

The corresponding activity events for small factory are as follows:

MACH1: $(\alpha_1, 0, \infty)$ $(\beta_1, 1, 2)$ $(\lambda_1, 0, \infty)$ $(\mu_1, 1, \infty)$ $(\kappa_1, 0, \infty)$

MACH2: $(\alpha_2, 0, \infty)$ $(\beta_2, 1, 1)$ $(\lambda_2, 0, \infty)$ $(\mu_2, 1, \infty)$ $(\kappa_2, 0, \infty)$

Let the set of forcible events be $\Sigma_{for} = \{\alpha_1, \alpha_2, \mu_1, \mu_2, \kappa_1, \kappa_2\}$ and the set of prohibitible events be equal to the set of forcible events ($\Sigma_{pro} = \Sigma_{for}$).

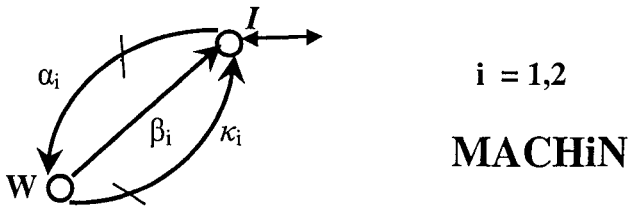


Figure 4.6: Example 4.1. Normal model of machines 1 and 2

The models of the DDM to be used are as follows:

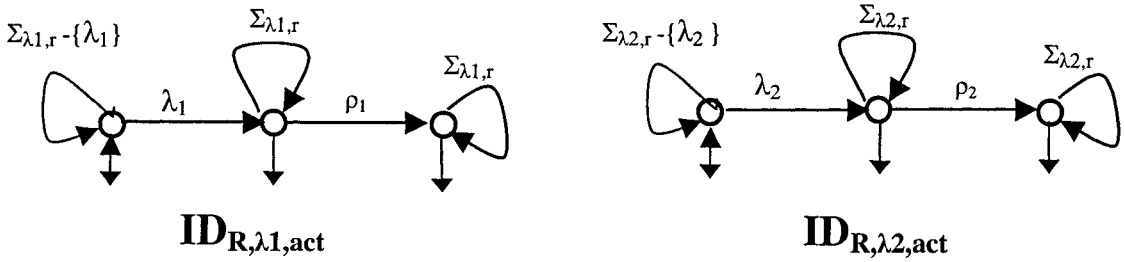


Figure 4.7: Example 4.1. Modular DDM for recovery mode

where λ_1 and λ_2 are breakdown faults respectively for machines 1 and 2; ρ_1 and $\rho 2$, the detection events corresponding to λ_1 and λ_2, are two activity events with $(\rho_1, 1, 3)$ and

(ρ_2, 1, 3). $\Sigma_{\lambda 1,r}$, $\Sigma_{\lambda 2,r}$, and ID_{Ract} (ID_{Ract} is the equivalent DDM) are as follows:

$$\Sigma_{\lambda 1,r} = \Sigma_p \cup \Sigma_f \cup \{\rho_2\} \cup \Sigma_r = \{\alpha_1, \alpha_2, \beta_1, \beta_2, \kappa_1, \kappa_2, \lambda_1, \lambda_2, \mu_1, \mu_2, \rho_2\}$$

$$\Sigma_{\lambda 2,r} = \Sigma_p \cup \Sigma_f \cup \{\rho_1\} \cup \Sigma_r = \{\alpha_1, \alpha_2, \beta_1, \beta_2, \kappa_1, \kappa_2, \lambda_1, \lambda_2, \mu_1, \mu_2, \rho_1\}$$

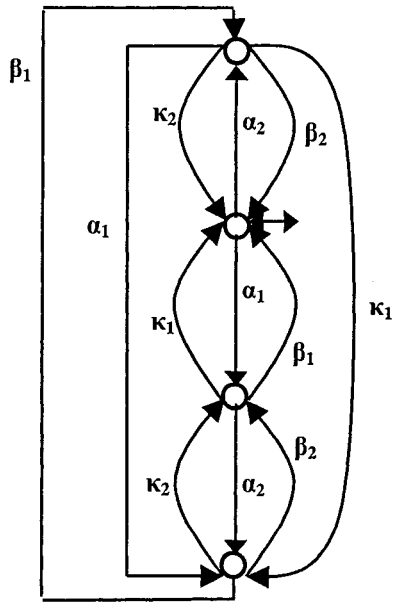
$$\mathbf{ID_{Ract} = comp(\ ID_{R,\lambda 1,act}\ ,\ ID_{R,\lambda 2,act}\ )}$$

Normal Mode: IGD_N

The ATG models of the machines in normal mode are illustrated in Fig. 4.6. Using the above description, the combined ATG and corresponding TDES models of the plant are constructed (Fig. 4.8).

$$IGD_{Nact} = \mathbf{comp}(MACH1N, MACH2N)$$

$$IGD_{N,\tau} = \mathbf{Timed\text{-}Graph}(\ IGD_{Nact})$$



IGD_{Nact}

Figure 4.8: Example 4.1. Equivalent ATG for machine 1 and 2 in normal mode

Normal-Transient Model: IGD_{NT}

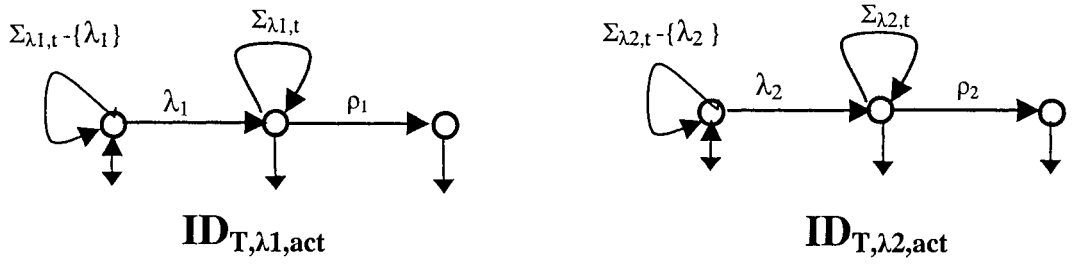
After including the fault events (λ_1 and λ_2) in the small factory, we obtain the ATG models of the machines used for computing GD_{NT} (Fig. 4.10).

The model of the diagnoser used in the construction of normal-transient model is obtained by removing selfloops in the detection state and removing recovery events from the event set in the $ID_{T,\lambda i,act}$. The resulting model is denoted by ID_{Tact}.

$$\Sigma_{\lambda 1,t} = \Sigma_p \cup \Sigma_f \cup \{\rho_2\} = \{\alpha_1, \alpha_2, \beta_1, \beta_2, \kappa_1, \kappa_2, \lambda_1, \lambda_2, \rho_2\}$$

$$\Sigma_{\lambda 2,t} = \Sigma_p \cup \Sigma_f \cup \{\rho_1\} = \{\alpha_1, \alpha_2, \beta_1, \beta_2, \kappa_1, \kappa_2, \lambda_1, \lambda_2, \rho_1\}$$

$$\mathbf{ID_{Tact} = comp(\ ID_{T,\lambda 1,act}\ ,\ ID_{T,\lambda 2,act}\ )}$$



$\mathbf{ID_{T,\lambda 1,act}}$ $\mathbf{ID_{T,\lambda 2,act}}$

Figure 4.9: Example 4.1. Modular DDM for transient mode

The normal-transient model of the system is the result of the following computations.

$$IG_{NTact} = \mathbf{comp}(\text{MACH1T, MACH2T})$$

$$\mathbf{IGD_{NTact} = comp(IG_{NT,act}\ ,\ ID_{Tact})}$$

$$IGD_{NT,\tau} = \mathbf{Timed\text{-}Graph}(\ IGD_{NTact})$$

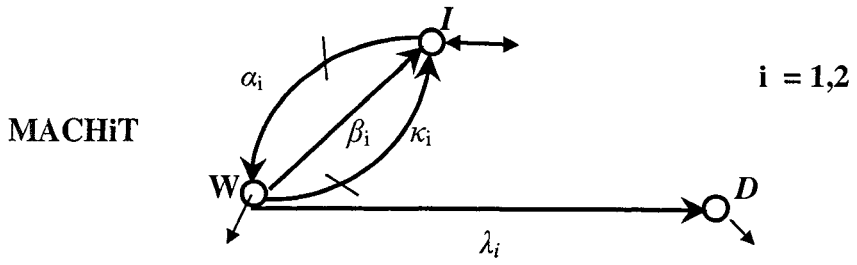


Figure 4.10: Example 4.1. Transient model of machines 1 and 2

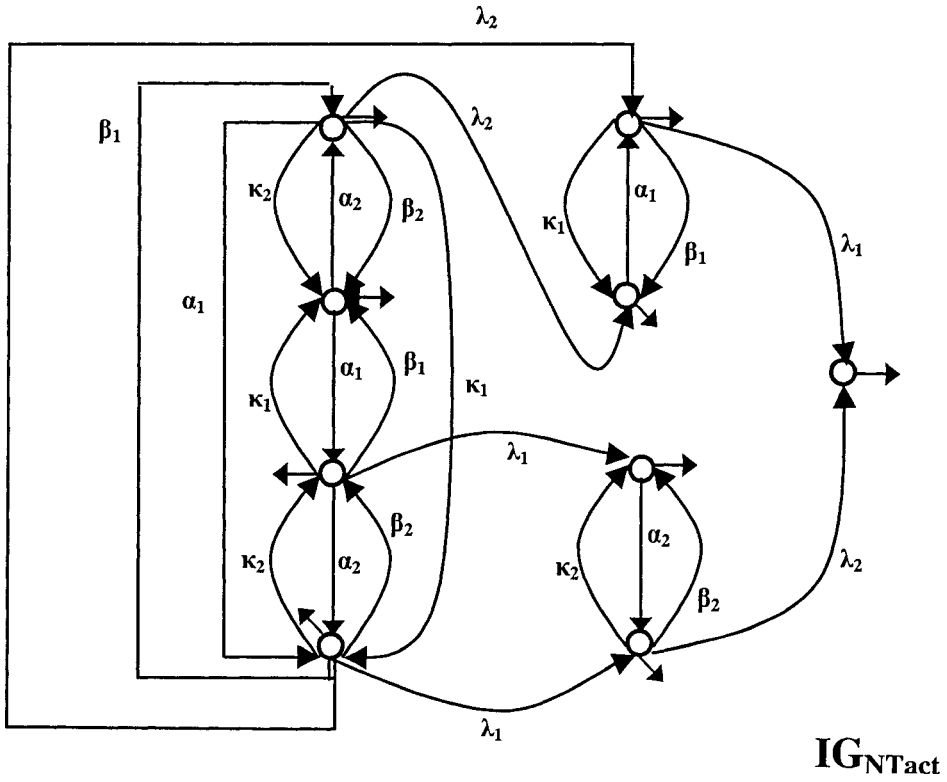


Figure 4.11: Example 4.1. Equivalent ATG for machines 1 and 2 in transient mode

Normal-Transient-Recovery Model: IGD_{NTR}

Using the same model shown in Fig.4.5 for the ATG models of the machines, we can obtain the ATG model of the plant (designated by IG_{NTRact}):

$$IG_{NTRact} = \textbf{comp}(MACH1R, MACH2R)$$

Fig. 4.12 illustrates the $\mathbf{IG_{NTRact}}$ where μ_1 , μ_2 are the recovery events.

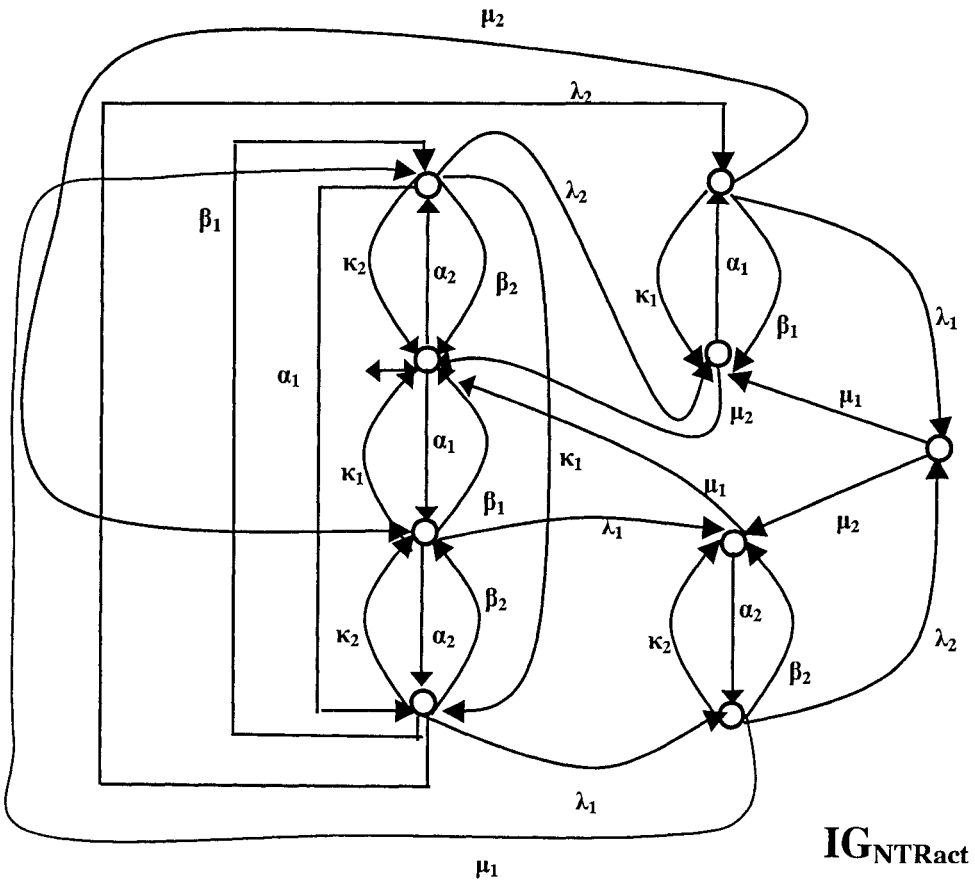


Figure 4.12: Example 4.1. Equivalent ATG for machine 1 and 2 in recovery mode

Finally, We have

$$\mathbf{IGD_{NTRact} = comp(\ IG_{NTRact}\ ,\ ID_{Ract})}$$

$$\mathbf{IGD_{NTR,\tau} = Timed\text{-}Graph(\ IGD_{NTRact})}$$

This example was given to illustrate modelling in our framework. A complete design example based on a workcell in a manufacturing line will be discussed in the next chapter.

4.2.5 Supervisor Design

In this section, we extend the untimed design procedure to the timed DES to construct supervisors for the three modes of the system. As before, we denote the normal, transient, and recovery supervisors by $\mathbf{S_N}$, $\mathbf{S_T}$, $\mathbf{S_R}$, respectively.

Normal Supervisor: S_N

Without loss of generality, we assume that in the normal mode, all of the events are observable and we use the **supcon** procedure to compute $\mathbf{S_N}$. If there are any unobservable events in the normal mode, then the theory of supervisory control of timed DES under partial observation can be applied.

Transient and Recovery supervisors: S_T, S_R

As the supervisors in the transient and recovery modes may not be able to observe the occurrences of fault events, the problem of finding a suitable controller for fault recovery are cases of supervisory control under partial observation. In order to synthesize a supervisor under partial observation using normal languages, we extend the LW procedure (chapter 2) to timed DES. The procedure steps are identical to the untimed case except for the difference in the concept of controllability in timed and untimed cases.

Let $G = (Q, \Sigma, \delta, q_o, Q_m)$ be a TDES and $P: \Sigma^* \rightarrow \Sigma_o^*$ be the natural projection defined over Σ. Now consider G_o to be defined over the alphabet Σ_o as an "observer's local model of G".

$$L_m(G_o) = P\,L_m(G) \qquad ; \qquad L(G_o) = P\,L(G)$$

The controllable and uncontrollable event subsets in G_o are:

$$\Sigma_{c,o} = \{\sigma \in \Sigma_c |\ \ P\,\sigma = \sigma\} = \Sigma_c \cap \Sigma_o$$

$$\Sigma_{u,o} = \Sigma_u \cap \Sigma_o$$

For $E_o \subseteq \Sigma_o^*$, let $C_o(E_o) = \{ K_o \subseteq E_o |\ K_o$ is controllable wrt $G_o\}$.

Let the specification language be $E \subseteq \Sigma^*$, and consider the following definitions.

$N_o = P\,sup\,N(E ; L_m(G))$,

$K_o = sup\,C_o(N_o)$

$J = P^{-1} K_o$

$K = L_m(G) \cap J$

In addition, as mentioned before, we assume that the tick event (τ) is observable. The following theorem is an extension of theorem 6.5.2 in [14]. Note that the timed version of SCOP is the same as its untimed version (with, of course, the difference in the concept of controllability in timed and untimed cases).

Theorem 4.1: Assume G is non-blocking, i.e. $\overline{L}_m(G) = L(G)$. If ($L_m(G)$, J) are nonconflicting and $K \neq \Phi$, then SCOP is solvable with $L_m(V/G) = K$.

Proof: First we show that K is normal and belongs to $S(E)$. The proof of this statement is similar to the proof given in [14] and we give the details here for completeness.

Let $N = sup\,N(E ; L_m(G))$. Then we see

$$K \subseteq L_m(G) \cap P^{-1}(N_o)$$

$$= L_m(G) \cap P^{-1}(PN)$$

$$= N \text{ (normality properties)}$$

$$\subseteq E$$

By definition

$$K = L_m(G) \cap P^{-1} K_o.$$

This means that K is *($L_m(G)$, P)-normal*, i.e.

$$K \in N(E ; L_m(G)).$$

Since $L_m(G)$ and J are nonconflicting,

$$\overline{K} = \overline{Lm(G) \cap J}$$

$$= \overline{Lm}(G) \cap \overline{J}$$

$$= L(G) \cap P^{-1} \overline{K}_o$$

i.e. $\overline{K}$ is (L(G), *P*)-normal. In other words $K \in \overline{N}(E;L(G))$.

In the second step, we show that K is controllable.

We have to show that for $s \in \overline{K}$

i. $\Sigma_{L(G)}(s) \cap \Sigma_u \subseteq \Sigma_K(s)$

ii. $\Sigma_K(s) \cap \Sigma_{for} = \Phi$ and tick $\in \Sigma_{L(G)}(s) \Rightarrow$ tick $\in \Sigma_K(s)$

To show (i), we verify that no uncontrollable events are disabled. Or for $\sigma \in \Sigma_u$, and $s\sigma \in L(G)$ then $s\sigma \in \overline{K}$. Again the proof of (i) is similar to the proof of controllability in [14]. We provide details for completeness.

If $P\sigma = \sigma$, then

$$(Ps)\sigma = P(s\sigma) \in P(L(G)) = L(G_o).$$

$Ps \in \overline{K}_o$ and from G_o-controllability of $\overline{K}_o$, we have $P(s\sigma) \in \overline{K}_o$ *i.e.*

$$s\sigma \in L(G) \cap P^{-1}(\overline{K}_o) = \overline{K}.$$

If $P\sigma = \varepsilon$, then

$$P(s\sigma) = Ps \in \overline{K}_o$$

which means

$$s\sigma \in L(G) \cap P^{-1}(\overline{K}_o) = \overline{K}$$

as required. Therefore, (i) is satisfied.

For (ii) we verify that the tick event cannot be disabled in the absence of forcible events. For simplicity, in the following, let τ denote tick. Suppose $s \in \overline{K}$ and $s\tau \in L(G)$. We have $P\tau = \tau$ and

$$\left.\begin{array}{l} s \in \overline{K} \\ s\tau \in L(G) \\ \Sigma_K(s) \cap \Sigma_{for} = \Phi \end{array}\right\} \rightarrow s\tau \in \overline{K}$$

By contradiction, we show that if $s\tau \notin \overline{K}$, then $\Sigma_K(s) \cap \Sigma_{for} \neq \Phi$.

Assume $s\tau \notin \overline{K}$ then

$$s\tau \notin P^{-1}\overline{K}_o$$

$$P(s)\tau \notin \overline{K}_o$$

Since

$$(Ps)\tau = P(s\tau) \in P(L(G)) = L(G_o)$$

From G_o-controllability of $\overline{K}_o$, there is $\alpha \in \Sigma_{for} \cap \Sigma_o$ such that

$$P(s)\alpha = P(s\alpha) \in \overline{K}_o$$

Also,

$$(Ps)\alpha = P(s\alpha) \in \overline{K}_o \subseteq L(G_o) = P(L(G))$$

Therefore

$$s\alpha \in P^{-1}P(L(G)).$$

Since $s \in L(G)$ and $\alpha \in \Sigma_{for}$,

$$s\alpha \in L(G)$$

In conclusion,

$$s\alpha \in L(G) \cap P^{-1}(\overline{K}_o) = \overline{K}$$

which implies $\Sigma_K(s) \cap \Sigma_{for} \neq \Phi$.

By (i) and (ii) K is controllable with respect to L(G).

The rest of the proof is similar to the untimed case. □

Based on the above theorem, we extend the LW method as follows:

0. **Given G, E, and the list NULL of P-unobservable events**
1. **GC = convert(G, tick, ξ)**
2. **EC = convert(E, tick, ξ)**
3. **NC = supnorm (EC , GC , NULL)**
4. **NCO = convert(NC, ξ, tick)**
5. **NO = project (NCO, NULL)**
6. **GO = project (G , NULL)**
7. **KO = supcon (GO , NO)**
8. **KODAT = condat (GO , KO)**
9. **PINVKO = selfloop (KO , NULL)**
10. **nonconflict (G , PINVKO)** = true **?**
11. **K = meet (G , PINVKO)**
12. **K** nonempty **?**

In the first and second steps, the procedure **convert** is used to exchange the tick event with an arbitrary uncontrollable event $\xi \notin \Sigma$. This is done so that supnorm procedure can be used in the presence of the tick event. We exchange ξ with the tick event at step four.

The construction of the proposed supervisor is performed in steps nine and ten. K will denote the plant under supervision if the answer to question twelve is true.

Remark 4.1: To investigate the effects of the normal supervisor (S_N) on the transient plant (GD_{NT}), we follow a procedure similar to that of the untimed case. Specifically, we verify the admissibility of S_N using the **condat** procedure.

$$\mathbf{STDAT = condat(GD_{NT}, S_{NT})}$$

If STDAT contains at least one state in which $\mathbf{S_{NT}}$ attempts to disable any uncontrollable event, then S_N should be disabled or modified (the complete procedure of modification is described in chapter 3). Furthermore, during the transient mode, if the tick is preempted by an unforcible event, then at that point, S_N must be disabled and be removed from the control loop.

4.2.6 Supervisor Verification

In order to verify the supervisor, we follow a procedure similar to that of the untimed case, except that care must be taken to ensure that, the tick event is not preempted except by forcible events. In other words, the marked behaviors of the plant under supervision of supervisors should be jointly coercive (chapter 2):

$L_m(\tilde{S}_{NT}/GD_{NTR})$, $L_m(S_R/GD_{NTR})$ must be jointly coercive with respect to GD_{NTR}.

Here, the procedure of construction $\tilde{S}_{NT}$ in TDES is similar to the un-timed DES.
The above condition can be verified using **condat**:

$$\mathbf{condat(\ meet(}\tilde{S}_{NT}, S_R), GD_{NTR})$$

To have a jointly coercive conjunction supervisor, the results of the above **condat** procedure must be acceptable. In other words, the tick event should not be prevented from occurring unless it is preempted by a forcible event.

It should be mentioned that since in the recovery mode only S_R controls the system, it is sufficient to check the joint coerciveness property for the normal-transient system (GD_{NT}). In other words, we may verify the following:

$L_m(S_{NT}/GD_{NT})$, $L_m(S_R/GD_{NT})$ must be jointly coercive with respect to GD_{NT}.

We use procedure **condat** to verify the above condition.

condat(**meet**(S_{NT} , S_R) , GD_{NT})

Chapter 5

An Illustrative Example: Manufacturing Cell

5.1 Overview

This chapter demonstrates the results from previous chapters (and in particular, partial observation and modular switching control) using a manufacturing system as an example. It provides an example of handling two simultaneous failures by reducing it to a single failure case (p=1). That is suitable for the cases where recovery procedures for faults are related.

We use an example from [13] and discuss modeling of the plant using TDES, building a DES model for the diagnosis system, and then computing the supervisors. Finally, we examine the supervisors in order to verify whether they are proper supervisors.

5.2 Manufacturing Cell

The Manufacturing Cell which, for simplicity we call MC, is shown in Fig. 5.1. This cell contains two numerically controlled machines (MACH1 and MACH2) and two conveyors (CONV1 and CONV2) as input and output ports of MC. CONV1 is used as an incoming infinite source of workpieces and CONV2 functions as an outgoing infinite sink. The machines take two parts from CONV1 and after processing, send them to CONV2.

There are two types of workpieces that each machine may process, p1 and p2. Transfer of parts between machines is considered as inside machine operations. Moreover, if a machine breaks down, then it may be repaired.

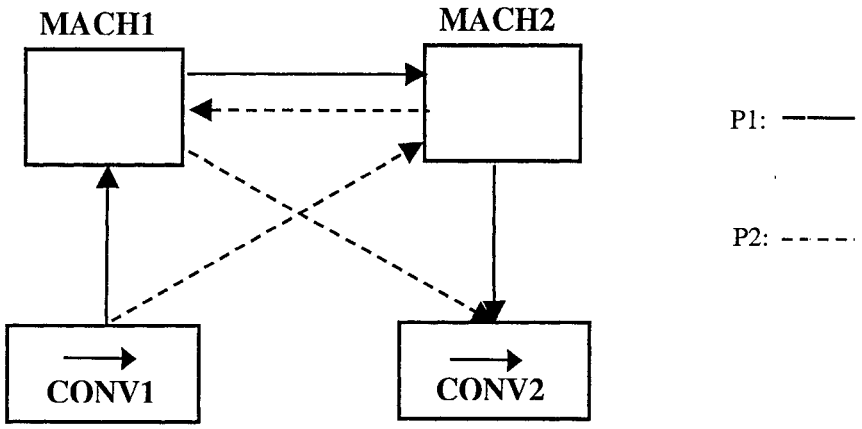
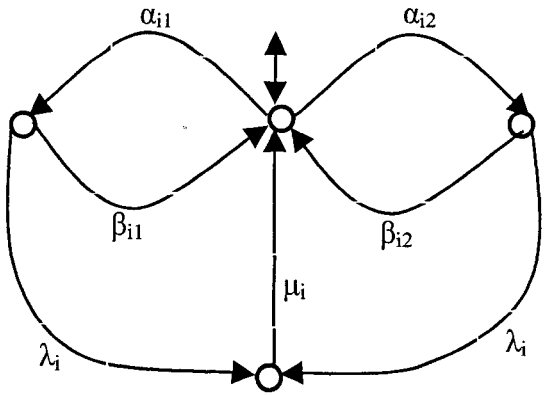
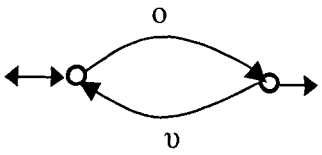


Figure 5.1: Manufacturing Cell

The ATG models of the machines (MACH1, MACH2) and the conveyors (CONV) are displayed in Fig. 5.2. Both of the conveyors are set to ON (or OFF) simultaneously.



MACHi, i = 1, 2



CONV

Figure 5.2: ATG models of machines and conveyors

The description of the events is as follows.

α_{ij} MACHi starts work on a P_j-part

β_{ij} MACHi finishes working on a P_j-part

λ_i MACHi breaks down

μ_i MACHi is repaired

o CONV1 and CONV2 are turned on

υ CONV1 and CONV2 are turned off

The following timed events are considered for MC.

MACH1: $(\alpha_{11}, 1, \infty)$ $(\beta_{11}, 3, 3)$ $(\alpha_{12}, 1, \infty)$ $(\beta_{12}, 2, 2)$

$(\lambda_1, 0, 3)$ $(\mu_1, 1, \infty)$

MACH2: $(\alpha_{21}, 1, \infty)$ $(\beta_{21}, 1, 1)$ $(\alpha_{22}, 1, \infty)$ $(\beta_{22}, 4, 4)$

$(\lambda_2, 0, 4)$ $(\mu_2, 1, \infty)$

CONV: $(o, 0, \infty)$ $(\upsilon, 1, \infty)$

The upper time bound of event λ_i could be any large number including ∞, but because breakdown may occur only when the machine is working, the upper time bound assigned to λ_i need not exceed the upper time bound β_{ij} for completion of the corresponding work cycle. Thus, for simplicity, we used the above values for the upper time bounds of the events λ_i.

The set of uncontrollable events ($\sum_{unc}$), forcible events ($\sum_{for}$), and prohibitible events ($\sum_{pro}$) are as follows:

$$\sum_{unc} = \{\lambda_i , \beta_{ij} , \rho_i \mid i, j = 1, 2 \}$$

$$\sum_{for} = \{ \alpha_{11}, \alpha_{12} , \alpha_{21}, \alpha_{22} , \mu_1 , \mu_2 , o , \upsilon \}$$

$$\sum_{pro} = \sum_{for}$$

Here, ρ_1 and ρ_2 are detection events which will be defined latter.

In each production cycle, the behavior of MC under control must satisfy the following specifications:

1. Logic-based specifications
   i. A given part can be processed by just one machine at a time.
   ii. A p1-part must be processed first by MACH1 and then by MACH2.
   iii. A p2-part must be processed first by MACH2 and then by MACH1.
   iv. One p1-part and one p2-part must be processed in each production cycle.
   v. If both machines are down, MACH2 is always repaired before MACH1.
   vi. Conveyors should be started prior to the machines.

2. Temporal specification
   i. In the absence of breakdown/repair events, a production cycle must be completed within 10 time units.

3. Quantitative optimality specification

   i. Subject to 2, production cycle time is to be minimized.

5.3 System Model

In this section, the activity transition graphs (ATG) as well as the timed transition graphs of the system in different modes (normal, transient, recovery) are discussed.

Plant

In this example, we have:

$$\sum_p = \{ \alpha_{11}, \alpha_{12}, \alpha_{21}, \alpha_{22}, \beta_{11}, \beta_{12}, \beta_{21}, \beta_{22}, o, \upsilon \}$$

$$\sum_f = \{\lambda_1, \lambda_2\}$$

$$\sum_d = \{\rho_1, \rho_2\}$$

$$\sum_r = \{\mu_1, \mu_2\}$$

Normal Mode

In the normal mode, the ATG model of the machines and conveyor are shown in Fig 5.3.

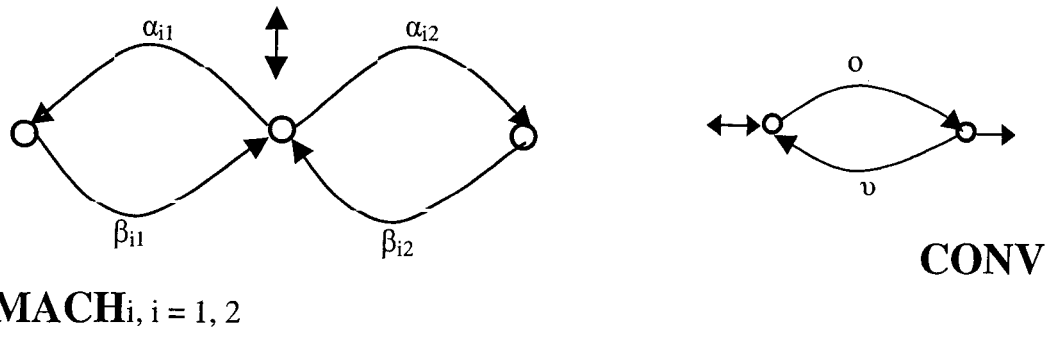


Figure 5.3: Machines and conveyors in normal mode

Applying TTCT composition procedure to these ATG's to get the equivalent ATG model, we have:

$$G_{Nact,1} = \textbf{comp}(\text{ NMACH1, NMACH2, CONV})$$

Normal-Transient Modes

In order to built the normal-transient ATG model of the system we compute

$$G_{NTact} = \textbf{comp}(\text{ TMACH1, TMACH2, CONV})$$

where TMACH1, TMACH2, and CONV (Fig. 5.4) are the ATG models of machine1, machine2, and conveyor, respectively, fully marked and without the repair events.

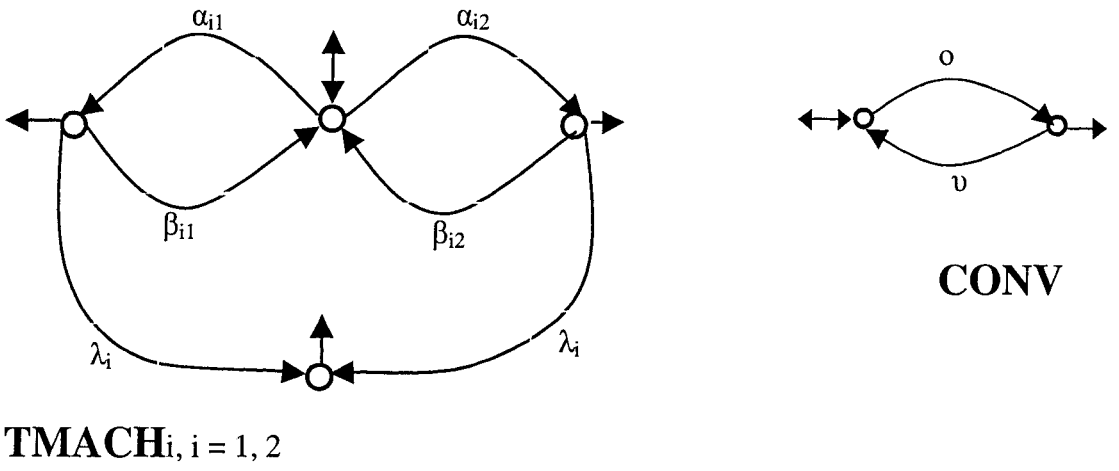
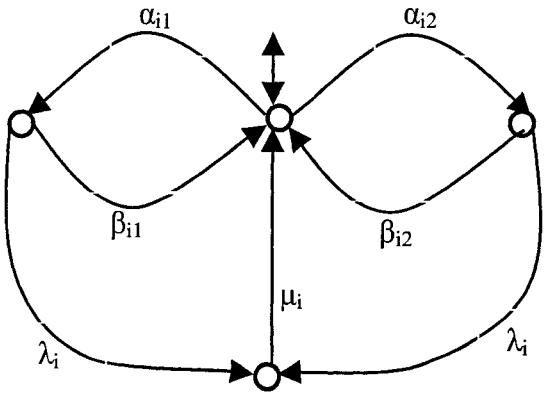


Figure 5.4: Machines and conveyors in transient mode

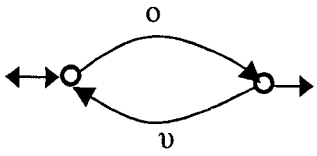
Normal-Transient-Recovery Modes

For this case, the ATG model can be constructed by applying the composition procedure to RMACH1, RMACH2, and CONV, which denote the ATG models of machine1, machine2, and conveyor, respectively. These models are displayed in Fig. 5.5.

G_{NTRact} = **comp**(RMACH1, RMACH2, CONV)



RMACHi, i = 1, 2



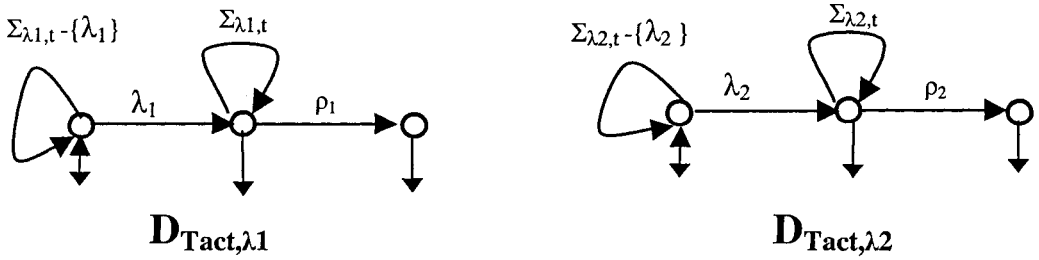
CONV

Figure 5.5: Machines and conveyors in recovery mode

DDM for the Diagnoser

The modules of the DDM's for the diagnoser (D_{Tact}) used for constructing the normal-transient model of the system are given in Fig. 5.6. ρ_1 and ρ_2 are the detection events corresponding to λ_1 and λ_2, respectively. We assume the following lower and upper time bounds for these events:

$$(\rho_1, 1, 3) \quad , \quad (\rho_2, 1, 3)$$



$D_{Tact,\lambda 1}$ $D_{Tact,\lambda 2}$

Figure 5.6: Modular DDM for the transient mode

In the above figure:

$$\Sigma_{\lambda 1,t} = \Sigma_p \cup \Sigma_f \cup \{\rho_2\} = \{ \alpha_{11}, \alpha_{12}, \alpha_{21}, \alpha_{22}, \beta_{11}, \beta_{12}, \beta_{21}, \beta_{22}, \lambda_1, \lambda_2, \rho_2, o, \upsilon \}$$

$$\Sigma_{\lambda 2,t} = \Sigma_p \cup \Sigma_f \cup \{\rho_1\} = \{ \alpha_{11}, \alpha_{12}, \alpha_{21}, \alpha_{22}, \beta_{11}, \beta_{12}, \beta_{21}, \beta_{22}, \lambda_1, \lambda_2, \rho_1, o, \upsilon \}$$

The required diagnoser model will be

$$\mathbf{D_{Tact} = comp(D_{Tact,\lambda 1}, D_{Tact,\lambda 2})}$$

The DDM for the diagnoser for the normal-transient-recovery mode (D_{Ract}) is designed as shown in Fig. 5.7 with

$$\Sigma_{\lambda 1,r} = \Sigma_p \cup \Sigma_f \cup \{\rho_2\} \cup \Sigma_r = \{\alpha_{11}, \alpha_{12}, \alpha_{21}, \alpha_{22}, \beta_{11}, \beta_{12}, \beta_{21}, \beta_{22}, \lambda_1, \lambda_2, \rho_2, \mu_1, \mu_2, o, \upsilon\}$$

$$\Sigma_{\lambda 2,r} = \Sigma_p \cup \Sigma_f \cup \{\rho_1\} \cup \Sigma_r = \{\alpha_{11}, \alpha_{12}, \alpha_{21}, \alpha_{22}, \beta_{11}, \beta_{12}, \beta_{21}, \beta_{22}, \lambda_1, \lambda_2, \rho_1, \mu_1, \mu_2, o, \upsilon\}$$

Finally,

$$\mathbf{D_{Ract} = comp(D_{Ract,\lambda 1}, D_{Ract,\lambda 2})}$$

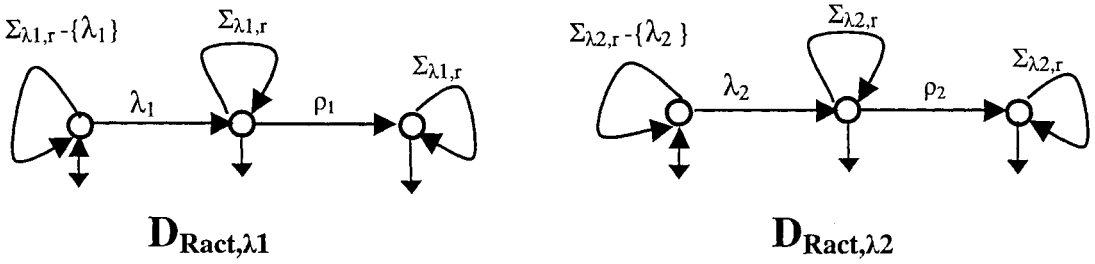
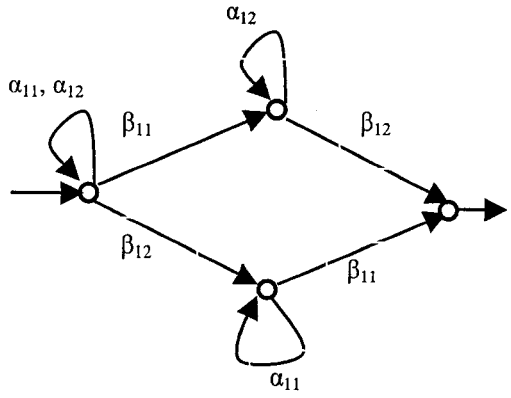
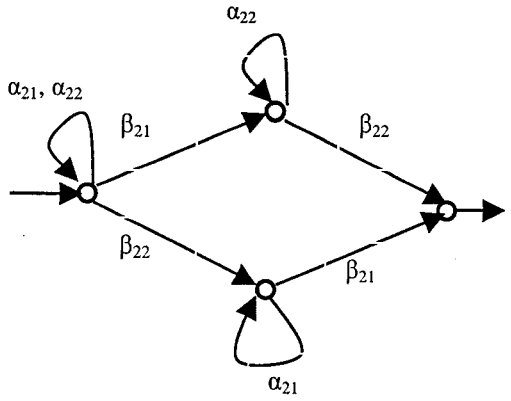


Figure 5.7: Modular DDM for the recovery mode

Remark 5.1: In this example, in order to ensure that the system under control completes only one cycle, we consider the following ATG's to be combined with the plant.



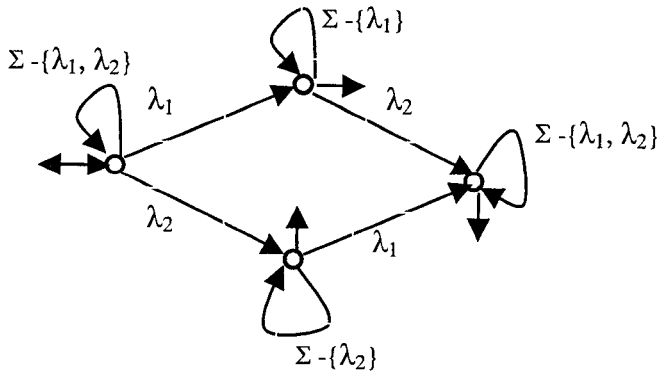
Limiter1



Limiter2

Figure 5.8: Limiter1 and Limiter2

In addition, for simplicity, we assume each fault can occur only once in a cycle. Therefore, we combine the following automaton with the entire plant under supervision (GD_{NTR}).



Limiter3

Figure 5.9: Limiter3

System to Be Controlled

In normal mode,

$$G_{Nact} = \mathbf{comp}(G_{Nact,1}, \text{Limiter1}, \text{Limiter2})$$

$$\mathbf{GD_{N,\tau} = Timed\text{-}Graph(GD_{Nact}) = Timed\text{-}Graph(G_{Nact})}$$

(States: 1202, Transitions: 2531)

The TDES models of the combined plant and the diagnoser in normal-transient are as follows:

$$GD_{NTact} = \mathbf{comp}(G_{NTact}, D_{Tact}, \text{Limiter1}, \text{Limiter2})$$

$$GD_{NT,\tau} = \mathbf{Timed\text{-}Graph}(GD_{NTact}) \quad ; \text{ (States: 3206, Transitions: 10010)}$$

It should be mentioned that since non-blocking is not among the transient specifications, in the construction of GD_{NTact}, all states of limiter1 and limiter2 are marked.

Finally, the normal-transient-recovery model of the system will be

$$GD_{NTRact} = \mathbf{comp}(G_{NTRact}, D_{Ract}, \text{Limiter1}, \text{Limiter2}, \text{Limiter3})$$

$$\mathbf{GD_{NTR,\tau} = Timed\text{-}Graph(GD_{NTRact})} \quad ; \text{ (States: 20918, Transitions: 67872)}$$

5.4 Specification Models

In this section, we formalize the specifications of each of the system modes.

Normal Specifications

1: A given part can be processed by just one machine at a time.

2: A p1-part must be processed first by MACH1 and then by MACH2.

3: A p2-part must be processed first by MACH2 and then by MACH1.

4: One p1-part and one p2-part must be processed in each production cycle.

5: Conveyors should be started prior to the machines.

6: In the absence of breakdown/repair events, a production cycle must be completed within 7 time units (production cycle time is to be minimized).

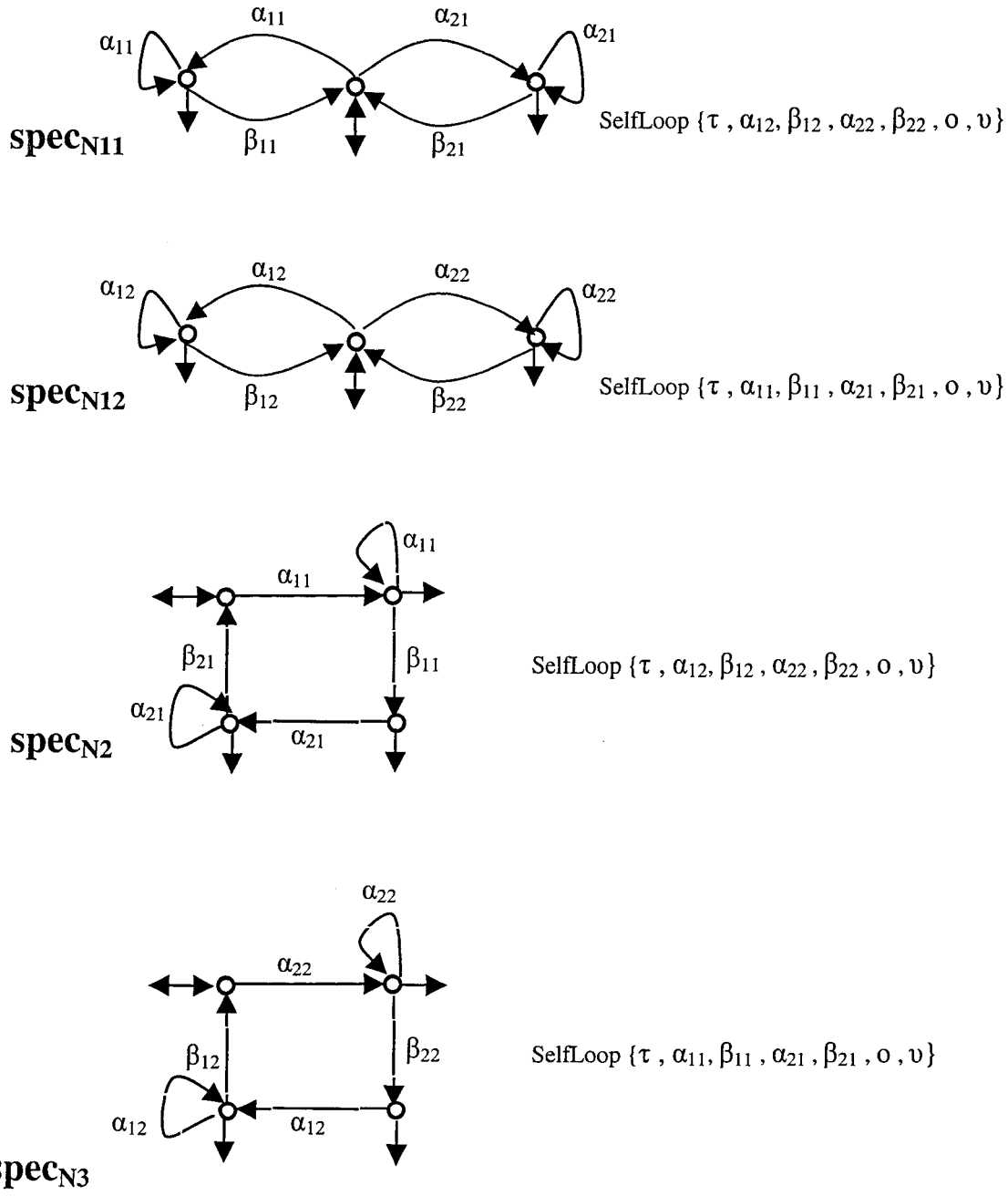
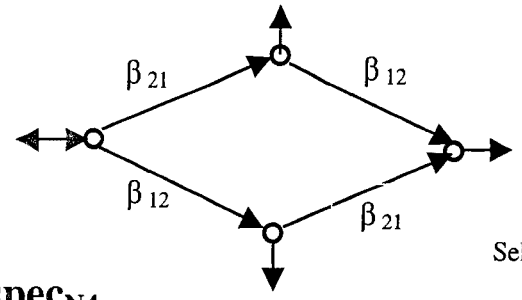
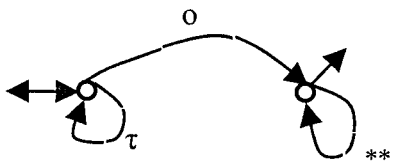


Figure 5.10: Normal specifications



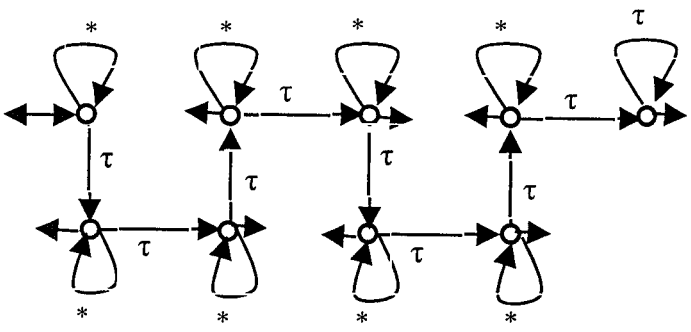
SelfLoop $\{\tau, \beta_{11}, \beta_{22}, \alpha_{11}, \alpha_{12}, \alpha_{21}, \alpha_{22}, o, \upsilon\}$

$spec_{N4}$



** = $\{\tau, \alpha_{11}, \alpha_{12}, \alpha_{21}, \alpha_{22}, \beta_{11}, \beta_{12}, \beta_{21}, \beta_{22}\}$

$Spec_{N5}$



$spec_{N6}$
(Timer 7)

* = $\{\alpha_{11}, \alpha_{12}, \alpha_{21}, \alpha_{22}, \beta_{11}, \beta_{12}, \beta_{21}, \beta_{22}, o, \upsilon\}$

Figure 5.11: Normal specifications (Cont'd)

Specifications 1, 2, and 3 are formalized as **$spec_{N11}$**, **$spec_{N12}$**, **$spec_{N2}$**, and **$spec_{N3}$**; the last three specifications are formalized as TDES **$spec_{N4}$**, **$spec_{N5}$**, and **$spec_{N6}$**, respectively.

As a result, the normal specification is the conjunction of the above specifications.

$E_{N,\tau}$ = meet($spec_{N11}$, $spec_{N12}$, $spec_{N2}$, $spec_{N3}$, $spec_{N4}$, $spec_{N5}$, $spec_{N6}$)

(States: 585, Transitions: 2001)

Transient Specifications

In this mode, the specifications are the following:

1: A given part can be processed by just one machine at a time.

2: A p1-part must be processed first by MACH1 and then by MACH2.

3: A p2-part must be processed first by MACH2 and then by MACH1.

4: One p1-part and one p2-part must be processed in each production cycle.

5: Conveyors should be started prior to the machines.

The TDES of the first four specifications, are obtained through the following selfloops:

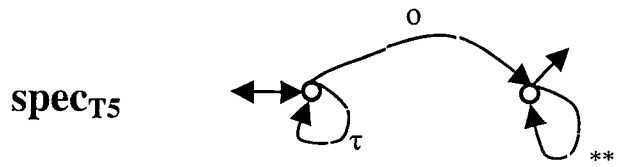
$\mathbf{spec_{T11}}$ = **selfloop(** $\mathbf{spec_{N11}}$, $\{ \lambda_1 , \lambda_2 , \rho 1, \rho 2\}$ **)**

$\mathbf{spec_{T12}}$ = **selfloop(** $\mathbf{spec_{N12}}$, $\{ \lambda_1 , \lambda_2 , \rho 1, \rho 2\}$ **)**

$\mathbf{spec_{T2}}$ = **selfloop(**$\mathbf{spec_{N2}}$, $\{ \lambda_1 , \lambda_2 , \rho 1, \rho 2\}$ **)**

$\mathbf{spec_{T3}}$ = **selfloop(**$\mathbf{spec_{N3}}$, $\{ \lambda_1 , \lambda_2 , \rho 1, \rho 2\}$ **)**

$\mathbf{spec_{T4}}$ = **selfloop(**$\mathbf{spec_{N4}}$, $\{ \lambda_1 , \lambda_2 , \rho 1, \rho 2\}$ **)**



** = $\{\tau , \alpha_{11}, \alpha_{12} , \alpha_{21}, \alpha_{22} , \beta_{11} , \beta_{12} , \beta_{21} , \beta_{22} , \lambda_1 , \lambda_2 , \rho 1 , \rho 2 \}$

Figure 5.12: The sixth transient specification

The transient specification is computed as

$$\mathbf{E_{T,\tau} = meet(\ spec_{T11}, spec_{T12}, spec_{T2}, spec_{T3}, spec_{T4}, spec_{T5})}$$

(States: 65, Transitions: 498).

Note that the transient specifications are more relaxed (or less restrictive) than the specifications of normal operation.

Recovery Specifications

The following are the recovery specifications:

1: A given part can be processed by just one machine at a time.

2: A p1-part must be processed first by MACH1 and then by MACH2.

3: A p2-part must be processed first by MACH2 and then by MACH1.

4: One p1-part and one p2-part must be processed in each production cycle.

5: Conveyors should be started prior to the machines.

6: If both machines are down, MACH2 is always repaired before MACH1.

7: After detection of faults in the system, the first command should be the shutdown of conveyors, next machine repairs, and then finishing the production cycle.

8: Production cycle must be completed in at most 30 time units.

The recovery specifications are more restrictive than transient specifications, since they involve some recovery control actions to accommodate faults. However, in this example, some of the recovery specifications are more relaxed compared with normal specifications. That is because the plant is faulty and it cannot work as efficiently as in the normal mode.

The TDES models of the first three specifications are formalized as follows:

$$\mathbf{spec_{R11} = selfloop(spec_{T11} , \{ \mu 1 , \mu 2\})}$$

$$\mathbf{spec_{R12} = selfloop(spec_{T12} , \{ \mu 1 , \mu 2\})}$$

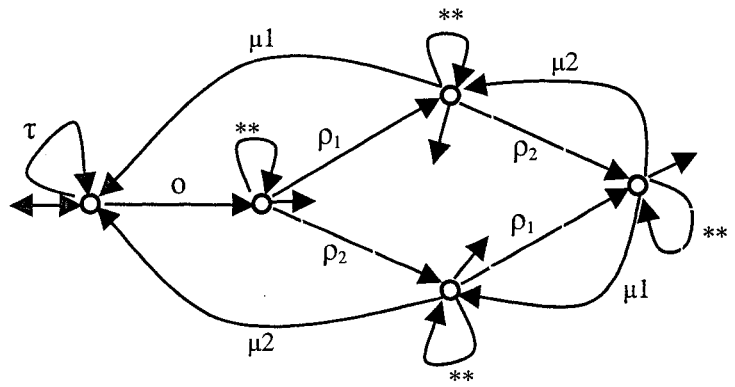
$$\mathbf{spec_{R2} = selfloop(spec_{T2} , \{ \mu 1 , \mu 2\})}$$

$$\mathbf{spec_{R3} = selfloop(spec_{T3} , \{ \mu 1 , \mu 2\})}$$

The fourth specification is constructed as:

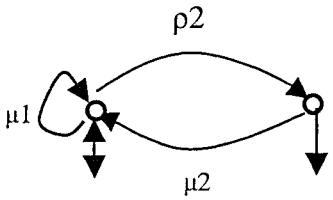
$$\mathbf{spec_{R4} = selfloop(spec_{T4} , \{ \mu 1 , \mu 2\})}$$

The TDES of the last four specifications are shown in figures 5.13 and 5.14.



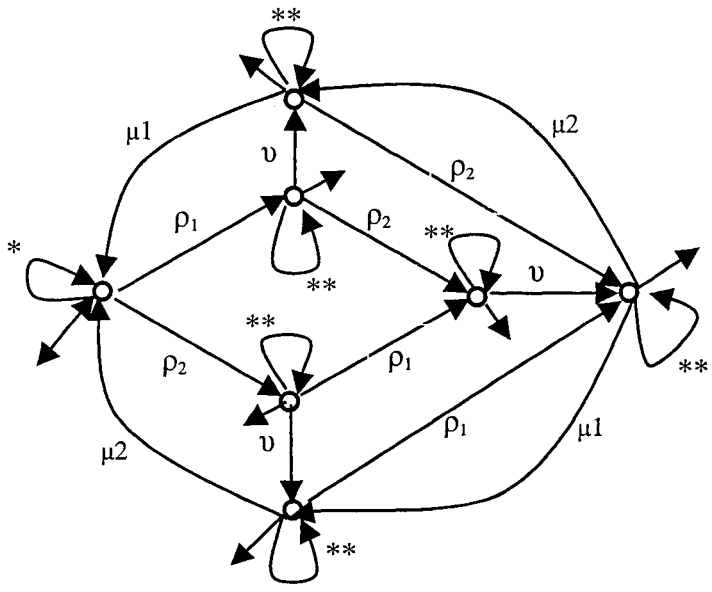
spec$_{R5}$

** = {τ , α_{11}, α_{12} , α_{21}, α_{22} , β_{11} , β_{12} , β_{21} , β_{22} , λ_1 , λ_2 , υ}



SelfLoop {τ , α_{11}, α_{12} , α_{21}, α_{22} , β_{11} , β_{12} , β_{21} , β_{22} , λ_1 , λ_2 , ρ1, o , υ}

spec$_{R6}$

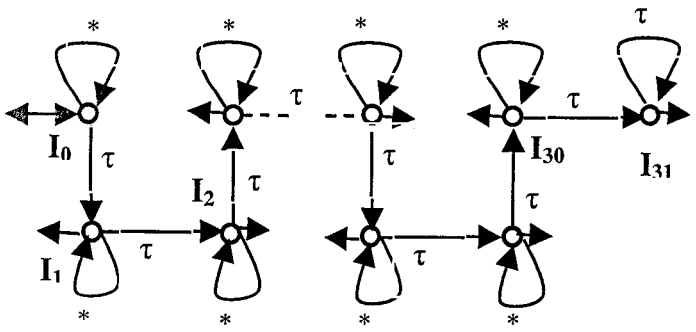


spec$_{R7}$

* = {τ , α_{11}, α_{12} , α_{21}, α_{22} , β_{11} , β_{12} , β_{21} , β_{22} , λ_1 , λ_2 , o}

** = {τ , β_{11} , β_{12} , β_{21} , β_{22} , λ_1 , λ_2 }

Figure 5.13: Recovery specifications



$Spec_{R8}$
(Timer 30)

$* = \{\alpha_{11}, \alpha_{12}, \alpha_{21}, \alpha_{22}, \beta_{11}, \beta_{12}, \beta_{21}, \beta_{22}, \lambda_1, \lambda_2, \rho_1, \rho_2, \mu 1, \mu 2, o, \upsilon\}$

Figure 5.14: Recovery specifications (cont'd)

Finally, we get

$$\mathbf{E_{R,\tau} = meet(spec_{R11}, spec_{R12}, spec_{R2}, spec_{R3}, spec_{R4}, spec_{R5}, spec_{R6}, spec_{R7}, spec_{R8})}$$

(States: 18432, Transitions: 95808)

5.5 Supervisor Design

In order to compute the supervisors in different modes, we start with the normal supervisor.

Normal Supervisor

The normal supervisor with respect to all the normal specifications is enforced by

$$\mathbf{S_{N,\tau} = supcon(GD_{N,\tau}, E_{N,\tau})} \quad ; \text{ (States: 25, Transitions: 29)}$$

Transient Supervisor ($S_{T,\tau}$)

The transient supervisor is computed as follows:

1. **TGC = convert($GD_{NT,\tau}$, tick, ξ)**
2. **TEC = convert($E_{T,\tau}$, tick, ξ)**
3. **TNC = supnorm(TEC , TGC , NULL)** , **NULL= $\{\lambda_1, \lambda_2\}$**
4. **TNCO = convert(TNC, ξ, tick)**
5. **TNO = project (TNCO , NULL)**
6. **TGO = project ($GD_{NT,\tau}$, NULL)**
7. **TKO = supcon (TGO , TNO)**
8. **TKODAT = condat(TGO , TKO)**
9. **$S_{T,\tau}$ = selfloop(TKO , NULL)**
10. **TK = meet ($GD_{NT,\tau}$, $S_{T,\tau}$)**
11. **TK** nonempty **?**

The computed supervisor ($S_{T,\tau}$) has 163 states and 722 transitions.

Remark 5.2: The normal-transient supervisor ($S_{NT,\tau}$) is the synchronous product of $S_{N,\tau}$ and $S_{T,\tau}$.

$$S_{NT,\tau} = \text{sync}(S_{N,\tau}, S_{T,\tau})$$

Following Remark 4.1, we compute

$$SNTDAT_{\tau} = \text{condat}(GD_{NT,\tau}, S_{NT,\tau})$$

It can be seen that $\mathbf{S_{NT,\tau}}$ tries to preempt the tick with non-forcible events in some states. This is done by $\mathbf{S_{N,\tau}}$. Modifying $\mathbf{S_{N,\tau}}$ will give us a new supervisor ($\mathbf{S_{NM,\tau}}$) which is an admissible supervisor for $\mathbf{GD_{NT,\tau}}$. Thus, the modified normal-transient supervisor is computed as follows:

$$\mathbf{S_{NTM,\tau} = sync(S_{NM,\tau}, S_{T,\tau})} \ ; \ \text{(States: 196, Transitions: 892)}$$

Here, we replace $\mathbf{S_{NT,\tau}}$ by $\mathbf{S_{NTM,\tau}}$ and use the same notation $\mathbf{S_{NT,\tau}}$ (in place of $\mathbf{S_{NTM,\tau}}$) for simplicity.

Recovery Supervisor ($\mathbf{S_{R,\tau}}$)

In this example, since the number of states and transitions are very large, we may run out of computer memory to perform the design. That is why we implement the following in two-step procedure.

First, we compute the supervisor $\mathbf{S_{R,1}}$ based on $\mathbf{GD_{NTR,\tau}}$ (the plant) and the conjunction of the first seven specifications ($\mathbf{E_{R,1}}$) (the specifications). Second, we compute the supervisor $\mathbf{S_{R,\tau}}$ based on $\mathbf{S_{R,1}}$ as the plant and the last remaining specification ($\mathbf{Spec_{R8}}$) as the specification.

It should be mentioned that in the case of untimed controllable sub languages, using this two steps procedure for computing supervisors gives an optimal supervisor [10]. However, in the case of this example (using controllable and normal sub languages), at least, we know that the obtained supervisor is sub optimal and satisfies the specifications.

The recovery supervisor is synthesized using the following procedure.

0. $E_{R,1}$ =meet($spec_{R11}$,$spec_{R12}$,$spec_{R2}$,$spec_{R3}$,$spec_{R4}$, $spec_{R5}$, $spec_{R6}$, $spec_{R7}$)
1. RGC1 = convert($GD_{NTR,\tau}$, tick, ξ)
2. REC1 = convert($E_{R,1}$, tick, ξ)
3. RNC1 = supnorm (REC1 , RGC1 , NULL) , NULL= $\{\lambda_1, \lambda_2\}$
4. RNCO1 = convert(RNC1, ξ, tick)
5. RNO1 = project (RNCO1 , NULL)
6. RGO1 = project($GD_{NTR,\tau}$, NULL)
7. RKO1 = supcon (RGO1 , RNO1)
8. $S_{R,1}$ = selfloop(RKO1 , NULL)
9. RGC2 = convert($S_{R,1}$, tick, ξ)
10. REC2 = convert($Spec_{R8}$, tick, ξ)
11. RNC2 = supnorm (REC2 , RGC2 , NULL) , NULL= $\{\lambda_1, \lambda_2\}$
12. RNCO2 = convert(RNC2, ξ, tick)
13. RNO2 = project (RNCO2 , NULL)
14. RGO2 = project($S_{R,1}$, NULL)
15. RKO2 = supcon (RGO2 , RNO2)
16. $S_{R,\tau}$ = selfloop(RKO2 , NULL)
17. GDNTRSRDAT = condat($GD_{NTR,\tau}$, $S_{R,\tau}$)
18. nonconflict($GD_{NTR,\tau}$, $S_{R,\tau}$) = true ?
19. RK = meet($GD_{NTR,\tau}$, $S_{R,\tau}$)
20. RK nonempty ?

The produced supervisor, which has 10401 states and 38844 transitions, is admissible and non-blocking with respect to $GD_{NTR,\tau}$.

5.6 Decoupling Condition

In this example, the decoupling condition is written as follows.

$$\mathbf{L}(\mathbf{S}_{NT,\tau} / \mathbf{GD}_{NT,\tau}) \subseteq \mathbf{L}(\mathbf{S}_{R,\tau} / \mathbf{GD}_{NT,\tau})$$

or

$$\mathbf{trim}(\mathbf{meet}(\mathbf{S}_{NT,\tau}/\mathbf{GD}_{NT,\tau}, \mathbf{complement}(\mathbf{S}_{R,\tau}/\mathbf{GD}_{NT,\tau}, \text{--}))) = \text{EMPTY ?}$$

where $\mathbf{S}_{NT,\tau}/\mathbf{GD}_{NT,\tau}$ and $\mathbf{S}_{R,\tau}/\mathbf{GD}_{NT,\tau}$ are the **meet** (product) of transient and recovery supervisors with the transient plant, respectively.

The reader can verify that the answer is "yes". In other words, the decoupling condition is satisfied for the manufacturing cell.

5.7 Supervisor Verification

The objective of this section is to verify whether the normal-transient and recovery supervisors are proper.

1. Admissibility property

For the normal-transient modes, the **condat** procedure is used to check the admissibility property:

$$\text{ISNT}\tau\text{GTDAT} = \mathbf{condat}(\mathbf{GD}_{NT,\tau}, \mathbf{S}_{NT,\tau})$$

ISNTτGTDAT illustrates that the supervisor is admissible with respect to $\mathbf{GD}_{NT,\tau}$.

2. Jointly coercive

In order to check this property, we find the result of the following **condat** procedure.

$$\text{SNTSRGDNTR} = \textbf{condat}(\ \textbf{meet}(\tilde{S}_{\text{NT},\tau}\ ,\ S_{\text{R},\tau})\ ,\ GD_{\text{NTR}})$$

SNTSRGDNTR does not include any state in which the conjunction supervisor must preempt the tick event with a non-forcible event. As a result, both supervisors are jointly coercive.

Note that we can also use the following procedure to verify if $\mathbf{S_{NT,\tau}}$ and $\mathbf{S_{R,\tau}}$ are jointly coercive:

$$\text{SNTSRGDNT} = \textbf{condat}(\ \textbf{meet}(S_{\text{NT},\tau}\ ,\ S_{\text{R},\tau})\ ,\ GD_{\text{NT}})$$

3. Non-blocking

Using **nonconflict** procedure to check this property, we find that the answer to the following question is "yes". In other words, the conjunction of $\mathbf{S_{NT,\tau}}$ and $\mathbf{S_{R,\tau}}$ is a non-blocking supervisor with respect to $\mathbf{GD_{N,\tau}}$.

$$\textbf{nonconflict}(\mathbf{GD_{N,\tau}}\ ,\ \textbf{meet}(\mathbf{S_{NT,\tau}}\ ,\ \mathbf{S_{R,\tau}})) = \text{true ?}$$

Now, since $\mathbf{S_{R,\tau}}$ is a non-blocking supervisor with respect to $\mathbf{GD_{NTR,\tau}}$, also $\mathbf{meet}(\mathbf{S_{NT,\tau}}\ ,\ \mathbf{S_{R,\tau}})$ is a non-blocking supervisor for $\mathbf{GD_{N,\tau}}$, we can conclude that fault recovery problem for the manufacturing cell is non-blocking (according to the Theorem 3.1, chapter 3).

In conclusion, the above verifications illustrate that the modular switching supervisor is a proper supervisor.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

In this thesis, a fault recovery scheme is developed based on discrete-event systems. We first present our recovery framework for untimed DES. We assume that the plant can be modeled as a finite state automaton. Also, we assume a dignosis system is available to detect and isolate unobservable faults with a finite delay. This diagnoser, generates a "detection" event in order to notify the supervisors about a fault occurring in the system. We use a finite state automaton to model detection delay of the diagnoser. This is essentially an abstraction of the diagnosis system.

The combination of the plant and diagnoser has three modes: normal, transient, and recovery. In general, there are different sets of specifications for each mode. We propose a modular switching approach for the synthesis of supervisor. We study the issues of non-blocking and supervisor admissibility. We also examine the effects of the controllers modules of each mode on other controllers and also on the plant in each mode.

We extend our scheme to timed discrete-event systems. Timing information can be used to improve the speed of diagnosis and, consequently, to improve the fault recovery. We introduce a timed model for the diagnoser and also present a framework for designing recovery procedures.

We illustrate our approach using various examples from process control and manufacturing systems. In particular, apply our technique to a manufacturing cell.

In this thesis, the fault diagnosis system can be any agent that informs the supervisors of the occurrence of faults in the system. No assumption is made about the methodology used to design the diagnoser. Thus, the technique studied in this thesis is not limited to any particular type of fault. It can potentially handle fault recovery problems in all types of applications in the automated systems such as autonomous systems and industrial control systems.

In this thesis, the diagnosis and control problems are separated and decoupled that allows simpler solutions for both problems. Also, use of a modular approach contributes to the simplicity and transparency of the control solution.

This approach is useful for systems that have built-in redundancy or systems that after recovery are allowed to continue with a degraded performance or shutdown completely. Our approach is also useful for the cases in which the faulty component can be repaired and the system as a result, can return to its normal mode.

We have not addressed transient faults. More study should be performed to deal with the recovery problems involving this type of fault. Furthermore, in this research, we have not studied the faults induced by operational changes (For example, changing the plant control philosophy, or changing the goals of production line).

6.2 Future Work

In the following, we describe some topics for the continuation of our research:

- In our thesis, we assumed that faults are not simultaneous. However, in practice, several faults can occur at the same instance. An external controller is needed to supervise the plant and its control system (controllers) and to give the necessary information to the sub controllers and also to set up a priority-based scheduling for recovering from faults.

- We assume that faults are the results of component failure and we model them as unobservable events in the system. What if the sequence of events has been changed not as the result of faults but as a consequence of operational changes? The operational

changes may include a change in objective criteria or a change in the production plan. This problem also can be considered as a problem of handling dynamic specifications or, at a higher level, can be called failure accommodation. One practical way of applying our approach to this problem is to define a new controllable event in the system, and enabling that event to perform a switch to the desirable supervisor which in turn, will apply a control logic to confine the plant behavior to the new set of specifications.

• Extending our recovery framework to decentralized discrete-event systems is another subject for future work. Here, the question is how to extend the approach to cases where we have a complex system with several subsystems that should be controlled separately (or with some interaction among subsystems)

• A software could be developed for the synthesis of recovery procedure. Two important issues are as follows. First, for large-scale systems, the computational complexity involved in using the RW theory will be a significant challenge that should be dealt with. Second, efficient user-friendly software procedures should be developed.

• In this study, our approach to fault recovery was linguistic and event-based and the aim of the supervisory control was to restrict the uncontrolled plant behavior in order to generate a desirable sublanguage. Another way to study the same concept is to use a state-based approach to the problem. Thus, the goal of the controllers would be to restrict the state of the plant to a set of desirable states. The main idea is that the controllers should prevent the plant from getting into those states that could lead to the violation of

some given constraints. An advantage of this method is that in this setup, it might be easier to deal with the problem of initiating supervisors after recovery since the control action would be a function of plant state.

• Another potential research topic is to extend our methodology to hierarchical discrete-event systems. Using a hierarchical DES approach, could improve the speed of active fault diagnosis. Applying hierarchical schemes to the fault recovery problem, in our framework, may improve the fault recovery procedure by reducing the complexity of the control problems.

• In this thesis, we did not study transient faults. Transient faults start at a particular time, remain in the system for some period and then disappear. This kind of fault can be caused by radiation particles e.g. by high-energy neutrons in aircrafts at high altitudes, or in spacecrafts by heavy-ions. In dealing with transient faults, the control system has to be more conservative when switching from normal to recovery controller. This problem requires further work.

Bibliography

[1] K. H. Cho and J. T. Lim, "*Synthesis of fault-tolerant supervisor for automated manufacturing systems: a case study on photolithographic process*," IEEE Trans. Robot. and Automat., vol. 14, no. 2, pp. 348-351, Apr. 1998.

[2] Brian C. Williams and P. Pandurang Nayak, "*A model-based approach to reactive self-configuration systems*," In Procs. of AAAI-96, pp. 971-978, Cambridge, Mass., 1996. AAAI, AAAI Press.

[3] D. Gordon and K. Kiriakidis, "*Design of adaptive supervisors for discrete event systems via learning*," in Proc. of the ASME Dynamic Systems and Contr. Division, International Mechanical Eng. Congress and Exposition, (Orlando, FL), Nov. 2000.

[4] D. Gordon and K. Kiriakidis, "*Supervisory of multiple-robots systems*," Proceedings of the American control conference, Arlington, vol. 3, pp. 2117 -2120 , VA June 25-27, 2001.

[5] S. Hashtrudi Zad, "*Fault diagnosis in discrete-event and hybrid systems*," Ph.D. thesis, Dept. of ECE, University of Toronto, 1999.

[6] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen and D. Teneketzis, "*Diagnosability of discrete-event systems*," IEEE Trans. Automat. Contr., vol. 40, no. 9, pp. 1555-1575, 1995.

[7] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen and D. Teneketzis, "*Failure diagnosis using discrete-event models*," IEEE Trans. Contr. Syst. Technology, vol. 4, no. 2, pp. 105-124, 1996.

[8] M. Sampath, S. Lafortune, and D. Teneketzis, "*Active diagnosis of discrete-event systems*," IEEE Trans. Automat. Contr., vol. 43, no. 7, pp. 908-929, 1998.

[9] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen and D. Teneketzis, "*Failure diagnosis of dynamic systems: an approach based on discrete event systems*," Proceeding of the American Control Conference, Arlington, VA June 25-27, 2001.

[10] P. J. Ramadge and W. M. Wonham, "*The control of discrete event systems*," IEEE, Proc., vol. 77, no. 1, pp. 81-89, January 1989.

[11] W. M. Wonham and P. J. Ramadge, "*Modular supervisory control of discrete event systems*," Maths. of Control, Signals and System, vol. 1, no. 1, pp. 13-30, 1988.

[12] B.A. Brandin and W.M. Wonham, "*Supervisory control of timed discrete-event systems*," IEEE Trans. Automat. Contr., vol. 39, no. 2, pp. 329-342, 1994.

[13] B.A. Brandin, W.M. Wonham, B. Benhabib "*Manufacturing cell supervisory control – a modular timed discrete-event system approach,*" IEEE Proc. Robot. and Automat., vol. 1, pp. 846-851, 1993.

[14] F. Lin, W. M. Wonham, *"On observability of discrete-event systems,"* Information Sciences 44 (2), pp. 173-198, 1988.

[15] F. Lin, W. M. Wonham, *"Supervisory control of timed discrete-event systems under partial observation,"* IEEE Trans. Automat. Contr., vol. 40 no. 3 pp. 558-562, 1995.

[16] M. Lawford, W. M. Wonham, *"Equivalence preserving transformations for timed transition models,"* IEEE Trans. Automat. Contr., pp. 1167-1179, vol. 40, no. 7, 1995.

[17] R. Kumar, V. K. Garg and S. I. Marcus, *"Predicates and predicate transformers for supervisory control of discrete event dynamical systems,"* IEEE Trans. Automat. Contr., pp. 232-247, vol. 38, no. 2, 1993.

[18] S. L. Chung, S. Lafortune, *"Limited lookahead policies in supervisory control of discrete event systems,"* IEEE Trans. Automat. Contr., pp. 1921-1935, vol. 37, no. 12, 1992.

[19] R. Kumar and M. Fabian, *"On supervisory control of partial specification arising in protocol conversion,"* Discrete Event Dynamical Systems: Theory and Applications.

[20] R. Guo, *"Fault diagnosis in a water treatment plant using discrete-event models,"* Master of engineering project report, Dept. of ECE, Concordia University, 2002.

[21] F. Lin, *"Robust and adaptive supervisory control of discrete event systems,"* IEEE Trans. Automat. Contr., pp. 1848-1852, vol. 38, no. 12, 1993.

[22] *TTCT* Software Package,

http://www.control.utoronto.ca/people/profs/wonham/wonham.html

(Current June, 2003)

[23] W. S. Lee, D.L. Grosh, F.A. Tillman and C.H. Lie, "*Fault tree analysis, methods, and applications- A review,*" IEEE Trans. Reliability, pp. 194-203, vol. R-34, no. 3, 1985.

[24] F. Lin, "*Diagnosability of discrete event systems and its applications,*" Discrete Event Dynamic Systems, pp. 197-212, vol. 4, no. 12, 1994.

[25] L.E. Holloway and S. Chand, "*Time templates for discrete event fault monitoring in manufacturing systems,*" Proc. Amer. Contr. Conf., Baltimore, MD, pp. 701-706, June 1994.

[26] W. M. Wonham, "*Notes on control of discrete-event systems,*" Systems Control Group, Edward S. Rogers Sr. Dept. of Elec. and Comp. Eng., University of Toronto, 1999.

[27] P. J. Ramadge and W. M. Wonham, "*Supervisory control of a class of discrete event processes,*" SIAM Journal on Contr. and Optimization, vol. 25, no. 1, pp. 206-230, January 1987.

[28] S. Hashtrudi Zad, R. H. Kwong and W. M. Wonham, "*Fault diagnosis in discrete-event systems: Framework and model reduction*," IEEE Trans. Automat. Contr., vol. 48, no. 7, pp. 1199-1212, July 2003.

[29] R. Isermann, "*Supervision, fault-detection and faul-diagnosis methods - An introduction,*" Contr. Eng. Practice, vol. 5, no. 5, pp. 639-652, 1997.