A General Review, Graphical Environment, and Applications for Discrete Event and Hybrid Systems in Robotics and Automation.

Tarek M. Sobh, Mohamed Dekhil, Peter-Pike Sloan, and Jonathan Owen[1]

UUCS-94-031

Department of Computer Science
University of Utah
Salt Lake City, UT 84112 USA

November 15, 1994

Abstract

In this report we present a review for the development of a theory for analyzing and predicting the behavior of discrete event systems (DES). Discrete Event Dynamic Systems (DEDS) are dynamic systems in which state transitions are caused by internal, discrete events in the system. DEDS are attracting considerable interests, current applications are found in manufacturing systems, communications and air traffic systems, robotics, autonomous systems, and artificial intelligence.

We also present an overview for the development of a graphical environment for simulating, analyzing, synthesizing, monitoring, and controlling complex discrete event and hybrid systems within the robotics, automation, and intelligent system domain. We start by presenting an overview of discrete event and hybrid systems, and then discuss the proposed framework. We also present two applications within the robotics and automation domain for such complex systems. The first is for formulating an observer for manipulating agents, and the second is for designing sensing strategies for the inspection of machine parts.

[1]This work was supported in part by NSF grant CDA 9024721, and a University of Utah Research Committee grant. All opinions, findings, conclusions or recommendations expressed in this document are those of the author and do not necessarily reflect the views of the sponsoring agencies.

Contents

List of Figures

List of Tables

1 Introduction

The underlying mathematical representation of complex computer-controlled systems is still insufficient to create a set of models which accurately captures the dynamics of the systems over the entire range of system operation. We remain in a situation where we must tradeoff the accuracy of our models with the manageability of the models. Closed-form solutions of mathematical models are almost exclusively limited to linear system models. Computer simulation of nonlinear and discrete-event models provide a means for off-line design of control systems. Guarantees of system performance are limited to those regions where the robustness conditions apply. These conditions may not apply during startup and shutdown or during periods of anomalous operation.

Recently, attempts have been made to model low and high-level system changes in automated and semi-automatic systems as discrete event dynamic systems (DEDS). Several attempts to improve the modeling capabilities are focused on mapping the continuous world into a discrete one. However, repeated results are available which indicate that large interactive systems evolve into states where minor events can lead to a catastrophe. Discrete event systems (DES) have been used in many domains to model and control system state changes within a process. Some of the domains include: Manufacturing, Robotics, Autonomous Agent Modeling, Control Theory, Assembly and Planning, Concurrency Control, Distributed Systems, Hierarchical Control, Highway Traffic Control, Autonomous Observation Under Uncertainty, Operating Systems, Communication Protocols, Real-Time Systems, Scheduling, and Simulation.

A number of tools and modeling techniques are being used to model and control discrete event systems in the above domains. Some of the modeling strategies include: Timed, untimed and stochastic Petri Nets and State Automata, Markovian, Stochastic, and Perturbation models, State Machines, Hierarchical State Machines, Hybrid Systems Modeling, Probabilistic Modeling (Uncertainty Recovery and Representation), Queuing Theory, and Recursive Functions.

The focus of this report will be to review the DES literature and some techniques used in the DEDS field, present a new framework for hybrid DES controllers and modelers, and discuss two application problems within the robotics and automation domain for which discrete event and hybrid systems play a significant role in the solution.

2 DES General Review

This section contains a description of a fairly complete and detailed bibliography of published books, journal, transaction, and conference papers, technical reports and other research publications related to the Discrete Event Dynamic Systems (DEDS) area.

The main objective of the section is to provide researchers with a comprehensive list of research in DEDS. Moreover, this section may serve, we hope, as a "look–up table" for literature review purposes, as well as to support the interested researcher in pursuing his/her interest in DEDS and/or in trying to identify new research topics.

The list of references has been divided into subject–indexed areas. Survey and/or general papers may have been listed in more than one subject areas, according to their specific content. All

Petri Net related publications, as applied to CIM, FMS and in general automated manufacturing systems, have been listed separately, mainly due to the fact that Petri Net related research is a rapidly evolving and expanding area.

Each subject–indexed references represent the state–of–the–art developments in the pertinent area. The authors believe that any finer area distinction will not serve a better purpose. This is left to the individual reader.

Due to the obvious space limitations, only selective references are listed in each of the subject area. The complete list of approximately 1,600 references is available to every reader from the ftp–site which contains the complete BibTeX file. Instructions related to how to access this file are given in Section 3.

It is very important to emphasize that a comprehensive list of written books is included in this section, while the representative 170 references, explicitly sited at the end of the section, represent only a "flavor" of the voluminous work reported in the literature and is by no means complete.

To facilitate reading the section, brief descriptions of reported work are presented in each subject–indexed section.

Finally, this work is part of the Discrete Event Systems Technical Activity Committee of the IEEE Robotics and Automation Society. The Committee's goal is to provide an annual, updated comprehensive list of references in DEDS, thus creating a dynamic database available to everyone through the ftp–site.

2.1 Subject Index

For each subject–index area a short description and several representative references are given.

A List of Representative Books

Good understanding of the DEDS area requires a knowledge of different topics, like control theory, probability, operation research, etc. Several books which provide a good introduction and a coverage of some advanced topics are listed in the sequel: [8], [9], [10], [17], [20], [23], [24], [25], [22], [27], [28], [29], [32], [30], [33], [35], [44], [54], [57], [58], [59], [61], [62], [64], [69], [71], [73], [74] [78], [80], [83], [85], [91], [96], [94], [98], [106], [108], [111], [112], [113], [114], [116], [119], [122], [127], [138], [140], [150], [147], [149], [153], [163], [165], [166], [167], [169], [170], [171], [173], [192], [199], [203], [208], [216], [222].

Algorithms

This subject includes description of various algorithms applied to DEDS problems, as well as the theory of algorithms in general. Most of these algorithms are related to the problems of scheduling, perturbation analysis, control and queues. Their definitions and a representative reference follows:

Scheduling: a sequence of actions is generated to achieve a desired goal [176],

Perturbation analysis: a system is analyzed by introducing a small change in the system parameters (perturbation) [95],

Control: selecting proper actions to achieve desired system behavior [191], and,

Queues: study of the stochastic nature of a system [81].

Applications

Various theoretical models (Petri Nets, stochastic, perturbation) are applied to particular problems like flexible manufacturing systems [5], resource sharing [45], transportation [143], real–time systems [207], etc. The models are used for analysis, synthesis, control and performance evaluation of a wide variety of DEDS.

Assembly

The assembly related references are divided into two main categories:

Plan generation: related to how to generate a sequence of actions or to schedule tasks to achieve desired functioning of the system [38],[209],[219], and,

Error recovery: related to how to recover from errors during the execution of a plan [110],[135].

A recently developed approach is based on the concept of "reverse assembly", widely referenced in recent studies.

Automation

This subject is closely related to assembly because the solution process for the problems of the plan generation and the error recovery involves automatization and generalization, i.e., for a given class of assembly processes a more or less general model (Petri Nets, hierarchical control) is used to automatically create plans and/or error recovery procedures. Representative references include [4], [16], [219], [221].

Concurrency

Concurrency includes coordination, control and error recovery of a set of concurrently competing processes. Real–time constraints, temporal semantics, resource sharing are some of the problems to be solved. Analysis, performance evaluation and verification of concurrent systems provide a way to measure the quality of the control process. Concurrency is not limited to the assembly and manufacturing systems, but it is also used in databases, distributed systems, protocols, etc. [120], [140], [155], [204].

Control

Various theoretical and practical aspects of the control of DEDS, like stability, robustness, controllability, etc. have been explored. This subject has the most references, due to the fact that the control is crucial to the system theory and therefore to DEDS [33], [61], [86], [116], [132], [139], [157], [213].

Hierarchical Control

Hierarchical control utilizes the structure of hierarchical, multilevel systems, and provides a more efficient way to control automated manufacturing systems by studying parts of the system at different level of abstraction, detail, and organizing the sequence of tasks to be executed in a hierarchical way [55], [92], [151], [217], [220].

Manufacturing

Manufacturing overlaps heavily with scheduling, automation and robotics. Various theoretical results are applied to this specific class of applications, e.g. design, analysis, performance evaluation, etc. [4], [42], [70], [97], [119], [162], [177], [206].

Markov Chains

The memoryless property of many real–time DEDS processes enables the application of Markov chains as a powerful tool for analysis. Various simulations, numerical methods, sensitivity analysis are applied to determine properties of the modeled DEDS [111], [166], [193], [211], [218].

Observers

As a part of sensitivity analysis and constructibility, observability is an important property of any DEDS. Observability is considered both during the design and implementation phase of a system [131], [158], [186], [202].

Operations Research

Methods of operation research are integrated with the system theory to provide solutions for problems like queuing networks, scheduling, etc. [2], [82], [88].

Perturbation Systems

Research reported in this area follows theoretical developments in perturbation analysis [41], [51], [78], [90], [185], [205].

Petri Nets

Petri Net models are extensively used in the modeling of DEDS. This includes ordinary Petri Nets, restricted Petri Nets, timed Petri Nets and high–level Petri Nets models. They provide a model for a very general DEDS. Properties of Petri Nets are used to determine or verify properties of the modeled system. Petri Nets and their modifications provide a very powerful mathematical and graphical tool for the study and evaluation of a wide class of systems. [1], [57], [66], [106], [107], [142], [222].

such extended state machine is used for system specification and modeling [31], [144], [215].

State Space

The state of the system at a time instant t represents a (measurable) behavior of the system. The set of all possible values that the state may take, the state space, is in general case very large. Problems of estimating the state–space size and developing new approaches to reduce the state–space or to deal with its complexity are described here [75], [145], [210].

Supervisors

Supervisors and supervisory controls refer to the form of control which involves enabling/disabling actions in a system. Such a control involves predictability, lookahead policies, languages generated by the control process, modularity, Petri Nets models, etc. [133], [160], [196], [201].

Survey Papers

Although, to the best of our knowledge, there is no survey paper which covers whole DEDS area, there are several survey papers covering some of the subjects. That includes Petri Nets, perturbation analysis, control architectures, scheduling theory, queuing networks, etc. [3], [19], [37], [60], [63], [65], [67], [89], [104], [109], [123], [124], [126], [136], [137], [154], [156], [178].

Theory

A good knowledge of topics related to the DEDS area, like control theory, probability, etc., require a good knowledge of the related theory. This is especially true for the system theory (control, hierarchical control, observers, supervisors), probability (Markov chains, queues, simulation), automata theory, etc. Selected papers and books provide a good introduction and as well as a coverage of advance level topics [34], [98], [138], [148], [153], [175].

2.2 How to Obtain the Complete Reference List

All references listed here can be reached by using ftp. The ftp–site is **swamp.cacs.usl.edu** and the references are stored in the directory **pub/deds** in the file **deds.bib** (BibTeX format). In addition, for each subsection in Section 2, there are three related files which contain selected references, one in the BibTeX format, one in the Postscript format, and one containing **\cite** commands to be used together with **deds.bib**. Reference which don't fit well in a single category are also listed. The **README** file contains information about all bibliography related files and how to use them.

An example of the ftp–session follows:

```
% ftp swamp.cacs.usl.edu
Connected to swamp.cacs.usl.edu.
220 swamp.cacs.usl.edu FTP server ...
Name (swamp.cacs:user): anonymous
```

```
331 Guest login ok, send ident as password.
Password:
230 Guest login ok, access restrictions apply.
ftp> cd pub/deds
250 CWD command successful.
ftp> get deds.bib
200 PORT command successful.
150 ASCII data connection for deds.bib ...
226 ASCII Transfer complete.
local: deds.bib remote: deds.bib
395377 bytes received ...
ftp> quit
221 Goodbye
```

If only a particular topic is needed, e.g., Petri Nets, selected references can be retrieved by getting following files:

petri.bib A list of selected references in the BibTeX format,

petri.ps A list of selected references in the Postscript format,

petri.cite A list of indexes of selected references (nocite commands), and,

petri.tex A LaTeX file used to generate **petri.ps**.

Therefore, for any of the subject areas, the related files are *area*.**bib**, *area*.**ps**, and *area*.**cite**.

2.3 Section Conclusion

The major objective of this subject–indexed bibliography section has been to summarize and group most research publications in the area of DEDS. As a result of this effort, it is believed that, through this extensive list, the present research directions related to several issues (modeling, synthesis, analysis, control, simulation, performance evaluation) of DEDS, will become more evident and clear. The reader will be able to isolated the most promising techniques related to the study of a specific aspect of an automated manufacturing system, or even define new directions in research.

The authors' wish is to serve continuously the professional community by providing on an annual basis an updated list of references, thus keeping track of current research and developments in this area.

3 Hybrid and Discrete Event Systems

Discrete event dynamic systems (DEDS) are dynamic systems in which state transitions are triggered by the occurrence of discrete events in the system. DEDS are suitable for representing hybrid systems which contains one or more of the following characteristics:

- continuous domain,
- discrete domain,
- chaotic behavior, and
- symbolic parameters.

Some examples of DEDS are:

Data Network: $A = \{send, receive, timeout, lost\}$

Shop with k jobs: $A = \{admit\_job, job\_finished\}$

Electric Distribution: $A = \{normal, short\_circuit, over\_current\}$

There are several frameworks can be used to model DEDS such as: finite automata, petri nets, Markov chains, etc. Choosing one of these frameworks depends on the nature of the problem being modeled and the implementation techniques available to implement this model. We will show some criteria for evaluating any framework.

DEDS has been applied to model many real-time problems and has been involved in different types of applications. Some of these applications are:

- Networks
- Manufacturing (sensing, inspection, and assembly)
- Economy
- Robotics (cooperating agents)
- Highway traffic control
- Operating systems

For more details about the DEDS applications see [18, 15, 87, 143, 187, 186, 184, 189, 188]. We believe that DEDS will have an important role in the development and improvement of many other applications in different disciplines.

In the following sections, we will investigate some of the different frameworks used to represent a DEDS model and the mathematical background and the notations used for each.

4 Discrete Event Models

As mentioned before, there are several representations and frameworks used in DEDS modeling. This section and the next *eight sections* are collected from various sources (papers and books) by different authors for the purpose of reviewing the DES area. Some of these frameworks are:

- Automata (untimed, timed, temporal, stochastic).
- Pushdown automata, μ-recursive, and Turing machines.
- Petrinets (timed, untimed).
- Markov chains.
- Queuing theory.
- Min-Max Algebra.
- Uncertainty modeling.
- Classical control.

These frameworks can be categorized in three different domains:

Timed vs. untimed models: the untimed models emphasize on the "state-event sequence" of a DEDS and ignores the holding time of each state, while the timed models, "time" is an essential part of the model.

Deterministic vs. probabilistic models: deterministic models assumes pre-knowledge of the sequence of events that will occur at any time, while probabilistic (stochastic) models associates probabilities with each event.

Computational model: which can be *logical models* in which the primarily questions are of qualitative or logical nature, while *algebraic models* can capture the description of the trajectories in terms of a finite set of algebraic operations. Finally, *performance models* are formed in terms of continuous variables such as average throughput, waiting time, etc.

Figure 1 shows the different models of representing DEDS and its characteristics. More about DEDS models can be found in [94].

4.1 Evaluation of DEDS Models

The evaluation of each frameworks can be done in four dimensions:

- Descriptive power
  - Language complexity

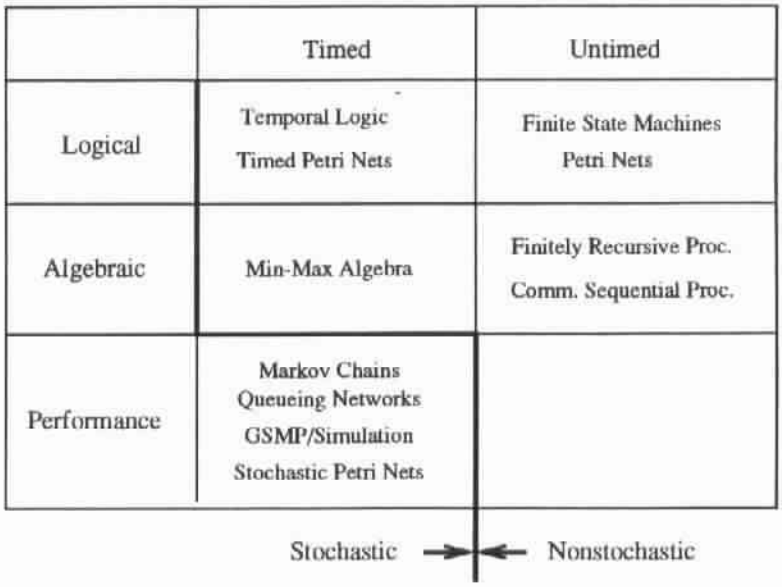


Figure 1: Different Models for DEDS

  - Algebraic complexity

- Implementation
- Performance evaluation
  - Logical correctness
  - real-time requirements
- Applications

Language complexity is based on the formal theory of languages. Each FSM generates a language L which is all possible traces of this FSM.

$$L(FSM) \subset L(Petrinets)$$

So, petri nets is more *language complex* than FSM.

Algebraic complexity is based on the systems theory. We can consider any algebraic system as a set of models and a set of operators that map one or more model to another. For example, In transfer functions, addition and multiplication reflect serial and parallel systems.

Logical correctness is a desirable property of the traces generated by any DEDS model the DES. For example, in the data network example, we must guarantee that each transmitted packet has been received correctly to the receiver.

Real-time requirements is a desirable property of the real-time response of the actual system. It is necessary to embed the DEDS model in a real-time environment.

5 Untimed Automata

In *untimed* models for DEDS, we only consider the sequence of state visited given a certain sequence of events. We don't consider *when* a state was visited or how long we stayed in a particular state. There are two main classes of untimed models, *state automata* and *Petri nets*. In this section we will consider state automata (in particular, finite state machines), as a framework for DEDS, and Petri nets will be discussed in Section 8.

5.1 Formalization and Notations

The DEDS model using finite state machine (FSM) is in the form:

$$G = (X, E, \Gamma, f, x_0)$$

where,

X: the finite set of states,
E: the finite set of possible events,
$\Gamma(x)$: the set of feasible or enabled events.
f: is the state transition function, $f : X \times E \longrightarrow X$.
x_0: is an initial state, $x_0 \in X$.
Figure 2 shows a simple FSM which can be described as:

$X = \{1, 2, 3\}$,

$E = \{a, b, c, d\}$,

$\Gamma(1) = \{a, c\}$,

$\Gamma(2) = \{a, b\}$,

$\Gamma(3) = \{d\}$,

$x_0 = 1$

and the state transition function f is represented by table 1.

	event			
state	a	b	b	d
1	2	–	3	–
2	3	1	–	–
3	–	–	–	3

Table 1: State transition function.

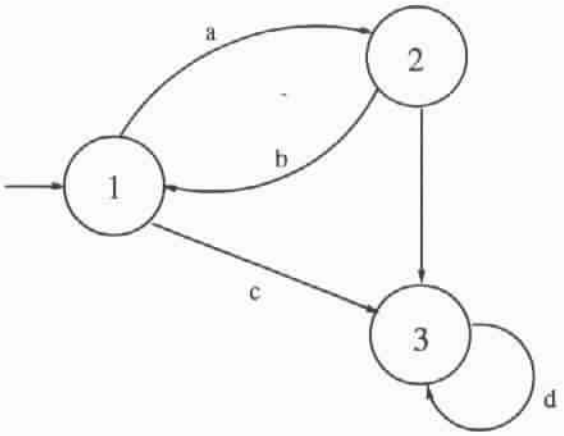


Figure 2: A Simple finite state machine.

5.2 Related Issues

There are some related issues to the use of state automata to represent DEDS models. Some of these issues are:

- Generated Language
- Pre-stability
- Stability
- f-Invariance
- Pre-stabilizability
- Stabilizability
- Observability
- WD Observability (With Delay)
- Indistinguishability

More information regarding these issues can be found in [145, 44, 146].

There is also an important issue related to that topic which is *output feedback stability* in which the goal is trying to *manipulate* the system's observer. It is the stabilization of the observer with state feedback. In this case we don't know when the system will enter the stable set. *Strong output stabilizability* in which we will know *when* the system will enter the stable set. A is strongly output E-stabilizable if there exists a state feedback K for the observer O such that O_K is stable with respect to $E_0 = \{x \in Z | x \subset E\}$.

6 Timed Models

In these models, time has an essential role in the representation and the question will include *when* an event happened, and not just did it happen or not. There are several frameworks to represent timed DEDS models. In this section we will discuss two of them, *timed automata* and *temporal automata.*

6.1 Timed Automata

In *timed* models for DEDS, the sequence of events is represented by a set of pairs of event and time. State sequence will be defined as:

$$\{(x_0, t_0), (x_1, t_1), \ldots)\}$$

where t_i denotes the time when x_i occurs (given t_0).

At time t_{k-1} event e_k is said to be *active*. The lifetime of an event e_k is:

$$\nu_K = t_K - t_{k-1}$$

A clock is set to ν_k at time t_{k-1} and starts ticking down till 0 (the lifetime expires for event e_k), then the event e_k occurs. *Active* and *occur* are equivalent to *enable* and *fire* in Petri nets.

For $t_{k-1} \leq t \leq t_k$,

$$y_k = t_k - t$$

$$z_k = t - t_{k-1}$$

A sample path is specified by the lifetime sequence $(V_1, V_2, \ldots)$.

The clock structure is a set associated with an event set E and is defined as:

$$V = \{v_i : i \in E\}$$

where,

$$v_i = \{\nu_{i,1}, \nu_{i,2}, \ldots\}$$

The clock structure will be the input to the DEDS machine to drive the state automata. To determine the next event we will use the function:

$$e_{k+1} = h(e_k, v_1, v_2, \ldots)$$

The timed state automaton Model will be:

$$M = (E, X, \Gamma, f, x_0, V),$$

where,

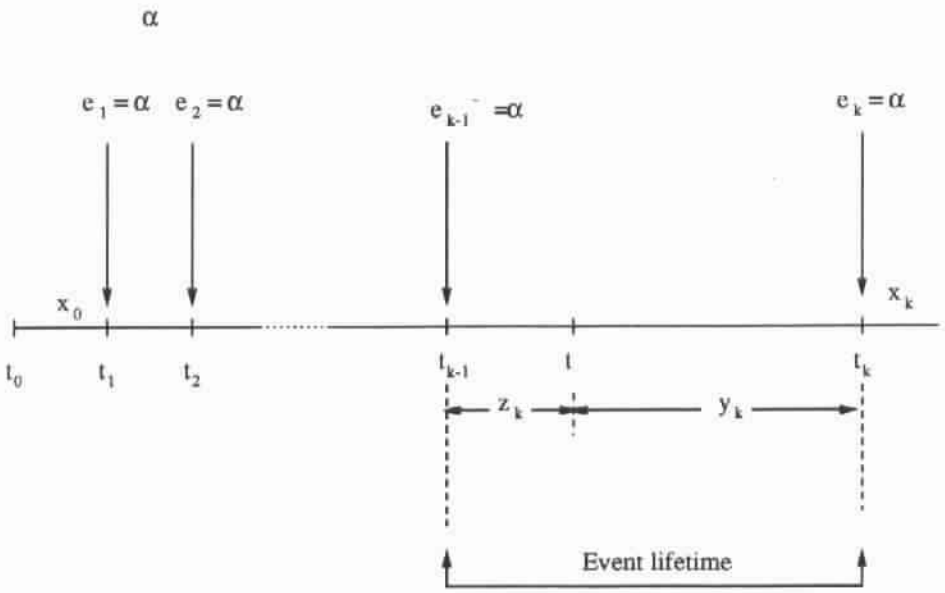


Figure 3: Sample Path of a DES with $E = \{\alpha\}$

E: a countable event set.
X: a countable state space.
$\Gamma(x)$: a set of feasible events for all $x \in X$ with $\Gamma(x) \subseteq E$.
f : a state transition function, $f : X \times E \to X$ defined only for $e \in \Gamma(x)$ when the state is x.
x_0: the initial state, $x_0 \in X$.
V: the clock structure.

Figure 3 shows an example of this formalism of timed automata where,

$E = \{\alpha\}$,

$\Gamma(x) = \{\alpha\}$ for all $x \in X$.

6.2 Temporal Automata

This is another timed model for DEDS. There are several distinctions of this model over the timed automata, such as:

- The environment is as important as the computing element.
- Composition and Decomposition of machines
- Machine hierarchy
- Representation of the duration of computation

First we will introduce some definitions and terminologies.

Time structure: a time structure is a set $T \subseteq R^+$ such that,

$0 \in T$ and,

for all $t \in R^+, sup\{x \in T | x \leq t\} \in T$

This means T is closed on its right.

Entities: an entity is a pair (T, D) where T is a time structure and D is a domain. So the entity can take set of states, set of events, set of inputs, or set of outputs.

Trace of an entity: trace of an entity is the values assigned to that entity over time. It is a function $v : T \rightarrow D$ defined only on the time structure of the entity. The set of all functions $T \rightarrow D$ will be denoted V_e.

Continuous time extension: sometimes we need at time t to know the last value taken by an entity. This requires the identification of the latest time of T which precedes t *The continuous time extension* of a trace v of the entity $e = (T, D)$ is the function $\bar{v} : R^+ \rightarrow D$ defined by:

$$\bar{v}(t) = v(sup\{x \in T | x \leq t\})$$

If $t \in T$ (the time structure of e) then $\bar{v}(t) = v(t)$, otherwise the value of $\bar{v}$ is the last value (in time) of v.

Causal functions: an entity e can be a function of other entities, as shown in table 2 The output can depend at time t on all the inputs from 0 to t. A *causal function* is a function f from the set of functions V_i over a set of entities $I = \{i_1, i_2, \ldots\}$ to V_e, and if we take two traces for e where e takes different values at time t then the input must differ at some time before i

	input		
state	α	β	γ
A	1	1	0
B	0	1	1
C	1	1	0
D	0	1	0

Table 2: An example of causal function.

Transductions: it is a tuple $(e, I, \delta, f_{init}, f)$ where

e: is an entity (T_e, D_e),

I: is a set of entities,

δ: is a nonnegative real,

f_{init}: is a function from $T_e \bigcap [0, \delta)$ to D_e,

f: is a function from $\prod_{i \in I} D_i$ to D_e

It defines a function F from $\prod_{i \in I} V_i$ to V_e :

$$F(v_{i1}, v_{i2}, \ldots)(t) = \begin{cases} f_{init}(t) & \forall t \in T_e | t < \delta \\ f(\bar{v_{i1}}(t-\delta), \bar{v_{i2}}(t-\delta), \ldots) & \forall t \in T_e | t \geq \delta \end{cases}$$

Causal systems: a causal system is a tuple (I, O, f) where,

I is a set of input entities,

O is a set of output entities, and

f is a causal function from $\prod_{i \in I} V_i$ to $\prod_{o \in O} V_o$.

The value of the entity e at time $t \in T_e$ is $v_i(t)$ if $e = i \in I$ and $f_o(v_{i1}, v_{i2}, \ldots)(t)$ if $e = o \in O$.

Temporal automata is a more specific form of causal systems.

Also, there are some important notes before we start with the formal definition of temporal automata.

- f_{init} defines the value of the entity before the entity has performed its first calculation.
- If $\delta = 0$ then no need for f_{init}.
- if $T_e \bigcap [0, \delta) = \{0\}$ then we need only $f_{init}(0)$.
- The function $\bar{v_i}(t)$ is a continuous function while $v_i(t)$ may not be continuous. This means that an entity doesn't have to wait for an input to happen at the input entity time structure, but it can take the last value of the input at any time (i.e no synchronization required.)
- As a convention we will say a transduction $(e, I, \delta, F_{init}, f)$ is from I to e, and if $\delta > 0$ then we call it *durational transduction.*
- A transduction $(e, I, \delta, f_{init}, f)$ where either $e \notin I$ or $\delta \neq 0$ defines a causal function from $\prod_{i \in I} V_i$ to V_e.

To illustrate some of these notations and definitions, we will show an example of a *time-out clock.* The clock keeps track of the time elapsed since the emission of a signal. The value of the clock is zero until it is activated, and from then on it increases in increments of ϵ, the clock grain size.

We will define the function f as follows:

$$f(start, clock) = \begin{cases} 0 & \text{if } clock = 0 \text{ and } start = 0 \\ clock + \epsilon & \text{otherwise} \end{cases}$$

We have a clock entity $clock = (N_\epsilon, R)$, where $N_\epsilon = \{k * \epsilon, k \in N\}$, and the transduction $(clock, \{start, clock\}, \epsilon, (0,0), f)$ When the entity *start* emits the signal, then the entity *clock* and the transduction will implement a clock giving the amount of time elapsed since the emission of the signal with a precision of ϵ. We will have:

$$v_{clock}(k * \epsilon) = f(\bar{v}_{start}((k-1) * \epsilon), v_{clock}((k-1) * \epsilon)).$$

According to the definition of f, as soon as v_{start} becomes 1, v_{clock} will start to increase by ϵ every ϵ seconds. Then in every history the value of the entity *clock* can be used to implement time-out or other time-dependent behavior.

A temporal automaton is a tuple (I, S, O, T) where,

I is a set of input entities,
S is a set of internal (states) entities,
O is a set of output entities,
T is a set of transductions with the following properties:
$\forall e \in S \bigcup O, T$ has a unique transduction to e and this transduction is from a set included in $I \bigcup S$
$\forall e \in S \bigcup O, \Delta(e)$ is finite and $e \notin \Delta^0(e)$, where $\Delta(e)$ is the set of entities that e depends on.

Any deterministic one-tape Turing machine can be simulated by a temporal automata.

In this framework a complicated system can be modeled as the composition of several modules, each module is represented by a temporal automata. The interaction between them are represented by the notion of a *connection*. the connection can be viewed as the wiring of the output of a machine to the input of another machine (see Figure 4).

A connection from one entity e_1 to an entity e_2 is denoted by $e_1 \triangleright e_2$ or $e_2 \triangleleft e_1$ and is a causal function f from V_1 to V_2 which associates to $v_1 \in V_1, v_2 = f(v_1)$ defined by $\forall t \in T_{e2}, v_2(t) = \bar{v_1}(t)$.

From this definition the time structure of e_1 and e_2 need not be equal (a connection does not imply synchronization.)

As an example of using temporal automata to model DEDS, let's consider modeling a mobile robot as follows:

A machine M will be used to implement the robot behavior "move forward in the middle of the corridor". This robot has two sensors modeled by the two machines $M_{l-sonar}$ and $M_{r-sonar}$. Assume the same time structure for the three machines which is: $N_\delta = \{k * \delta, k \in N\}$ where $\delta = 50$ milliseconds.

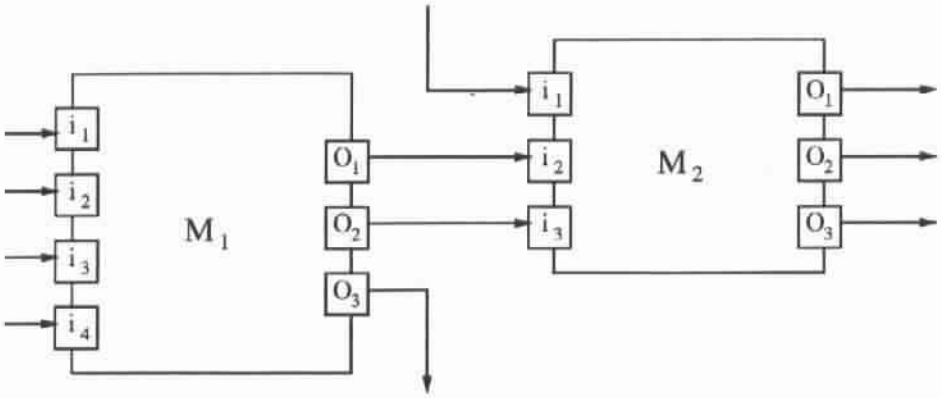


Figure 4: Connecting two temporal automata.

The sonar is very simple, with one input entity *measure* with domain $(0, 1)$ and one output entity *distance* with domain $\in R^+$. The transduction to *distance* is $(distance, \{measure\}, \delta, (0,0), f)$ with $f(measure) =$ if $measure = 1$ then function-sonar() else 0.

The robot M has three inputs d_l, d_r from the sensors and i is the order dived to the robot. The domains of the inputs are:

$$D_{d_l} = D_{d_r} = R^+$$

$$D_i = \{move, stop\}$$

The robot M has three outputs m, o_l, o_r which are, order to the motor, order to the left sonar, and order to the right sonar. with the domains:

$D_{0_l} = D_{o_r} = \{0, 1\}$,
$D_m = \{forward, backward, left, right, stay\}$
The transduction to o_l is $(o_l, \{i\}, \delta, (0,0), g)$
The transduction to o_r is $(o_r, \{i\}, \delta, (0,0), g)$
The transduction to m is $(m, \{i, d_l, d_r\}, \delta, (0,0), h)$

where,

$g(i) = if\, i = move then 1 else 0$, and

$$h(i, d_l, d_r) = \begin{cases} stay & \text{if } i = stop \\ left & \text{if } i = move \text{ and } d_l - d_r > \Delta \\ forward & \text{if } i = move \text{ and } -\Delta \leq d_l - d_r \leq \Delta \\ right & \text{if } i = move \text{ and } d_l - d_r < -\Delta \end{cases}$$

with $\Delta = 20$ centimeters.

The wiring:

$W = \{o_r \triangleright measure_{r-sonar}, o_l \triangleright measure_{l-sonar}, distance_{r-sonar} \triangleright d_r, distance_{l-sonar} \triangleright d_l\}$

over $M, M_{l-sonar}$, and $M_{r-sonar}$ induce a temporal automaton (see [121]).

7 Stochastic Timed Automata

The stochastic clock structure associated with an event set E is a set of distribution functions:

$$G = \{G_i : i \in E\}$$

The stochastic clock sequence will be:

$\{V_{i,k}\} = \{V_{i,1}, V_{i,2}, \ldots\}, \qquad i \in E,$

$V_{i,k} \in R^+, k = 1, 2, \ldots$

Each event $i \in E$ has a distribution function denoted by G_i which describes the random clock sequence $V_{i,k}$

We will assume that the clock sequences are independent of each other. The distribution function is defined as:

$$G_i = P[V_i \leq t]$$

There are two types of uncertainties:

- Stochastic clock structure.
- The initial state is a random variable with the *pdf*:

$$p_0(x) = P[X_0 = x], \qquad x \in X$$

- The transition function is also probabilistically specified with the transition probability:

$$p(\acute{x}; x, \acute{e}) = P[\acute{X} = \acute{x} | X = x, \acute{E} = \acute{e}]$$

where $x, \acute{x} \in X, \acute{e} \in E$

Note that if $\acute{e} \notin \Gamma(x)$, then

$$p(\acute{x}; x, \acute{e}) = 0$$

A stochastic timed automata will be defined as a six tuple:

$$(E, X, \Gamma, p, p_0, G)$$

where,

E is a countable event set

X is a countable state space

$\Gamma(x)$: is a set of feasible or enabled events, defined for all $x \in X$

$p(\acute{x}; x, \acute{e})$: is a state transition probability

$p_0(x)$: is the pdf $P[X_0 = x]$

$G = \{G_i : i \in E\}$: is a stochastic clock structure.

8 Petri Nets

A formal definition of a Petri net is as follows:

$PN = (P, T, A, W)$

- $P = \{p_1, p_2, p_3, \ldots, p_n\}$
- $T = \{t_1, t_2, t_3, \ldots, t_m\}$
- $A = A_{in} \cup A_{out}$ with
  - A_{in} set of elements from $\{P \times T\}$
  - A_{out} set of elements from $\{T \times P\}$
- W = a weight function, $w : A \rightarrow \{1, 2, 3, \ldots\}$

P is a set of places, T a set of transitions, A a set of directed arcs of two forms, the input arcs, connect places to transitions, and the output arcs connect transitions to places, and W is the weight of the arcs. Graphically the places are represented by circles, the transitions by bars, and the arcs by arcs.

A *marking* x of a Petri Net is a function $x :\rightarrow P\{0, 1, 2, ...\}$

A marking defines a row vector $x = [x(p_1), x(p_2), \ldots, x(p_n)]$, where n is the number of places in the Petri Net.

A marked Petri Net is defined as:

$PN = (P, T, A, W, x_0)$

- where (P, T, A, W) is as above
- x_0 is an initial marking

The marking elements are called *tokens*, and are graphically represented by dots in there place.

A transition is *enabled* if all its input places have a marking greater than the input arcs weight.

A Petri net is executed by the firing of transitions. A transition may only fire if it is enabled. When a transition fires, it removes the weight of the input arcs from there corresponding places,

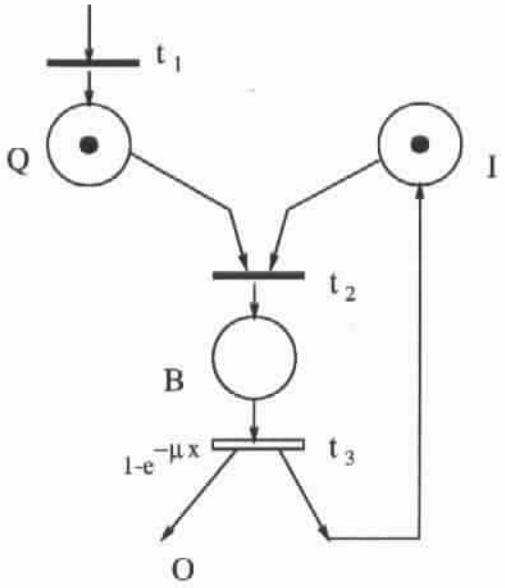


Figure 5: A simple Petri net model.

and adds the weight of the output arcs to there corresponding places. The number of input arcs does not have to equal the number of output arcs, and the total number of tokens does not have to stay constant. For our purposes the marking of a Petri net represents the current *state* of the system, and the firing of a transition represents an *event occurrence*.

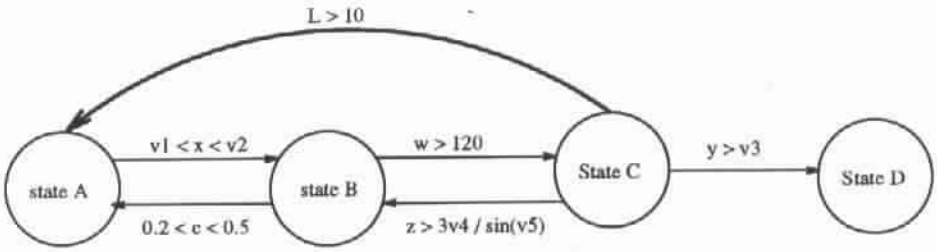
A timed Petri Net is a Petri net where each event has a firing delay associated with each transition, this is the delay between the enabling and firing of the transition. Stochastic Petri Nets, are timed Petri Nets where the delay is a random variable. The graphical representation of this has a thin bar if there is no delay, and a thick bar with its distribution next to it if there is. Figure 5 shows a simple Petri net model with untimed, timed, and stochastic places.

8.1 Continuous Petri Nets

The *continuous petri nets* (CPN) are used to model a DES systems that contains large number of states. In CPN the marking of a place is a real positive or null number. A maximum firing speed is associated with each transition. When a transition is fired, a quantity r is fired where r is a real number. So, the quantity of marks taken from or added to the places is defined by r and by the weights of the input and output arcs.

Two types of CPN have been defined in [56, 12]: *constant-speed continuous petri nets* (CCPN), and *variable-speed continuous petri nets* (VCPN). The CCPN is a model in which the firing speed vector remains constant during a pre-defined functioning intervals.

A modeling tool for controlled speed continuous petri nets called CSCPN is described in [76]. Using this tool, the maximum firing speed can be controlled and thus the firing speed vector is not constant.



trans. Variables	V1	V2	V3	V4	V5
Level 1	12	15	0.03	170	25
Level 2	10	12	0.07	100	35
Level 3	6	8	0.15	50	40

Figure 6: A Simple DRFSM

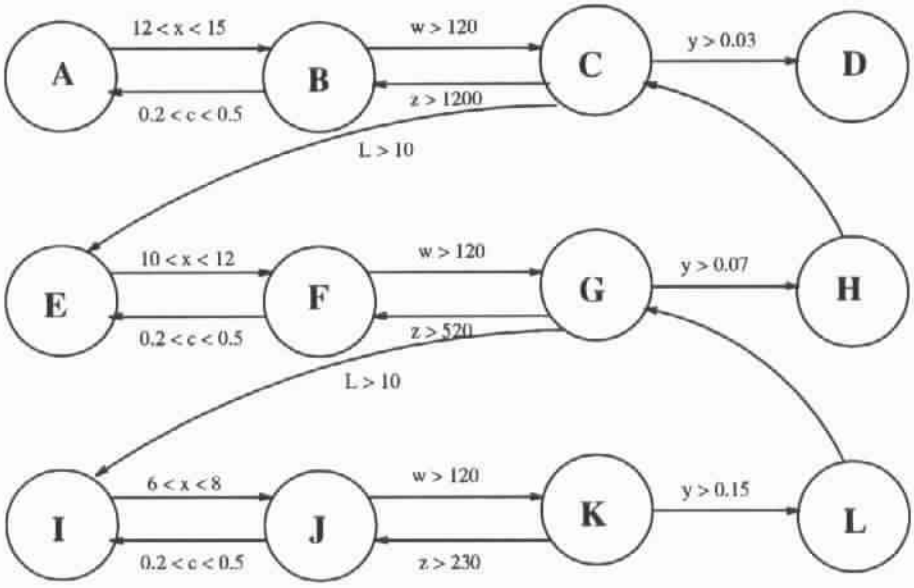


Figure 7: Flat Representation of a Simple DRFSM

10 Finite Recursive Process Models

Finite recursive processes (FRP) are a class of models suitable for systems constructed from interacting modules of discrete events such as manufacturing systems, communication networks, etc.

The basic characteristics of FRP are:

- The set of events can change dynamically
- Recursive equations are used to represent a process
- Termination of a process is formulated by modifying the process space
- Every Petri net can be represented as an FRP.

10.1 Definitions and Notations

To formally define a FRP, we will start by some definitions and notations. First we will have another definition for a FSM which will be:

$$M = (q_0, A, f, Q)$$

The transition function can be defined as:

$f(q, \langle\rangle) := q$

$$f(q, s\hat{}\langle a\rangle) := \begin{cases} \text{undefined, if } f(q,s) \text{ or } f(f(q,s),a) \text{ is undefined} \\ f(f(q,s),a), \text{ otherwise.} \end{cases}$$

If $f(q, s)$ is defined, then we say that M can execute the trace s from state q. The traces of M is the set of all sequences that M can execute from the initial state q_0.

$$\text{tr } M := \{s \in A^* | f(q_0, s) \text{ is defined}\}$$

A synchronous composition of FSMs can be defined as follows:

Let $A = A_1 \bigcup A_2$, the sequence $s \uparrow_{A_i^*}$ (read as projection of s on A_i^*) is defined as follows:

$$\langle\rangle \uparrow_{A_i^*} := \langle\rangle$$

$$s\hat{}\langle a\rangle \uparrow_{A_i^*} := \begin{cases} (s \uparrow_{A_i^*})\hat{}\langle a\rangle \text{ if } a \in A_i \\ s \uparrow_{A_i^*} \text{ if } a \notin A_i \end{cases}$$

The *synchronous composition* of $M1, M2$ is the machine $M1 \| M2$ defined as follows:

$$M1||M2 := (Q_1 \times Q_2, A, f, (q_{01}, q_{02}))$$

$$f((q_1, q_2), a) := \begin{cases} (f_1(q_1, a), f_2(q_2, a)) \text{ if } a \in A_1 \bigcap A_2 \\ (f_1(q_1, a), q_2) \text{ if } a \in A_1 \sim A_2 \\ (q_1, f_2(q_2, a)) \text{ if } a \in A_2 \sim A_1 \\ \text{undefined otherwise.} \end{cases}$$

$$trM = \{s \in A^* | s \uparrow_{A_i^*} \in trM_i, i = 1, 2\}.$$

Process Space $\prod$: a deterministic process P is a triple:

$$P = (trP, \alpha P, \tau P)$$

where,

- trP is the set of traces of P,
- $\alpha P(s)$ is the *event function* $trP \rightarrow 2^A$, and it specifies the traces that P can execute or block after it executes s
- $\tau P(s)$ is the *termination function* $trP \rightarrow \{0, 1\}$ and it specifies whether P will terminate or not after it executes s

Some conditions that should be satisfied by the process P

1. $\langle\rangle \in trP$,
2. $s\hat{}t \in trP \Rightarrow s \in trP$,
3. $s\hat{}\langle a\rangle \in trP \Rightarrow a \in \alpha P(s)$,
4. $\tau P(s) = 1 \Rightarrow s\hat{}t \notin trP$ unless $t = \langle\rangle$.

These definitions are also important for the formalization of FRP:

- Operations defined on processes are functions $\prod \rightarrow \prod$.
- A function $f : \prod \rightarrow \prod$ is *continuous* if for any increasing sequence of process $\{P_i, i \geq 0\}$, $f(P_i)$ is increasing and

$$f(\bigcup_{i \geq 0} P_i) = \bigcup_{i \geq 0} f(P_i).$$

- A *postprocess* P/s of a process P is defined for $s \in trP$ by:

$$tr(P/s) := \{t \in A^* | s^\frown t \in trP\},$$

$$\alpha(P/s)(t) := \alpha P(s^\frown t),$$

$$\tau(P/s)(t) := \tau P(s^\frown t).$$

- Let $P \uparrow n$ denote the subprocess containing only traces of P that have length at most n, where $n \geq 0$.

$$P = \bigcup_{n \geq 0} P \uparrow n.$$

$f : \prod \to \prod$ is **constructive** (con) if:

$$f(X) \uparrow n+1 = f(X \uparrow n) \uparrow n+1.$$

$f : \prod \to \prod$ is **nondestructive** (ndes) if:

$$f(X) \uparrow n = f(X \uparrow n) \uparrow n.$$

f is **strictly nondestructive** (sndes) if its is *ndes* but not *con*.

The transformation from a process to a FSM can be done as follows:

$$M_p = (\{P/s | s \in trP\}, A, f, P/\langle\rangle)$$

where f is defined by:

$$f(P/s, a) := \begin{cases} P/(s^\frown\langle a\rangle) & \text{if } s^\frown\langle a\rangle \in trP \\ \text{undefined} & \text{otherwise.} \end{cases}$$

$$trM_p = trP.$$

Theorem

$$X = f(X)$$

where $f = (f_1, \ldots, f_n) : \prod^n \to \prod^n$.

For any set of initial conditions:

$$X_i \uparrow 0 = Z_{0i}, i = 1, \ldots, n$$

that is consistent, i.e.,

$$Z_{0i} = f_i(Z_{01}, \ldots, Z_{0n}) \uparrow 0, i = 1, \ldots, n$$

This has a unique solution $X \in \prod^n$ satisfying the initial conditions, provided the following two conditions are fulfilled by f:

- $C1$: Each f_i is continuous and *ndes* in all its arguments.
- $C2$: f contains no sndes loop.

This enables us to define processes recursively. If we take $n = 1$ (scalar) then the solution will be:

$$X = \bigcup_{i \geq 0} f^{(i)}(Z_0)$$

where $f^{(i)} = f \circ f \ldots \circ f$ (i times).

10.2 Formalization of FRP

There are five operators that serve as the building blocks.

(1) Deterministic choice operator: given A_0, distinct elements $a_1, \ldots, a_n$, in A_0 and $\tau_0 \in 0, 1$, this operator maps $(P_1, \ldots, P_n) \in \prod^n$ into the the process Q denoted by:

$$Q = (a_1 \to P_1 | \ldots | a_n \to P_n)_{A_0, \tau_0}$$

Example:

$X = (a \rightarrow X | b \rightarrow SKIP_{\{a,b\}})_{\{a,b\},0}$

has a unique solution X with:

$trX = \{a^n, a^n b | n \geq 0\}$,

$\alpha X = \{a, b\}, \tau X(a^n) = 0$,

$\tau X(a^n b) = 1$.

(2) Synchronous Composition: the *synchronous composition* of processes P, Q is the process $P \| Q$ defined as follows:

$$\langle\rangle \in tr(P \| Q)$$

if $s \in tr(P \| Q)$, then $s^\frown\langle a\rangle \in tr(P \| Q)$ iff

- $s^\frown\langle a\rangle \uparrow_P \in trP, s^\frown\langle a\rangle \uparrow_Q \in trQ$
- $a \in \alpha P(s \uparrow_P) \bigcup \alpha Q(s \uparrow_Q)$

$\alpha(P \| Q) := \alpha P(s \uparrow_P) \bigcup \alpha Q(s \uparrow_Q)$

$$\tau(P \| Q)(s) := 1 \Leftrightarrow \begin{cases} \tau P(s \uparrow_P) = 1, \\ \text{ and } \tau Q(s \uparrow_Q) = 1; \text{ or} \\ \tau P(s \uparrow_P) = 1, \\ \text{ and } \alpha Q(s \uparrow_Q) \subset \alpha P(s \uparrow_P); \text{ or} \\ \tau Q(s \uparrow_Q) = 1, \\ \text{ and } \alpha P(s \uparrow_P) \subset \alpha Q(s \uparrow_Q) \end{cases}$$

$$\tau(P \| Q)(s) := 0, \ otherwise$$

(3) Sequential Composition: *Sequential composition* of processes P, Q is denoted by $P; Q$ and intuitively, $P; Q$ proceeds according to P until it successfully terminates ($\tau P(s) = 1$), and then it proceeds according to Q.

Example:

$$Y = (a \rightarrow Y; X | d \rightarrow SKIP_{\{\}A,0})$$

$$X = (b \rightarrow SKIP_{\{\}A,0})$$

gives

$$tr\ Y = \{a^n d b^n | n \geq 0\}.$$

This trace is not regular and cannot be generated by a FSM.

(4,5) Local and Global Change: These two operators change the event function of the process.

Let B, C be subsets of A. the *local change* operator corresponding to B, and C maps the process $P \in \prod$ into the process $P^{[-B+C]}$ defined as follows:

$$trP^{[-B+C]} := \{s \in trP | b \in B \Rightarrow b \text{ is not the first event of } s\}$$

$$\alpha P^{[-B+C]}(\langle\rangle) := \{\alpha P(\langle\rangle) \sim B\} \bigcup C, \text{ and}$$
$$\alpha P^{[-B+C]}(s) := \alpha P(s), \text{ for } s \neq \langle\rangle$$

$$\tau P^{[-B+C]}(s) := \tau P(s), \text{ for } s \in trP^{[-B+C]}.$$

The corresponding *global change* operator maps process P into $P^{[[-B+C]]}$ defined by:

$$trP^{[[-B+C]]} := \{s \in trP | b \in B \Rightarrow b \text{ is not in } s\}$$

$$\alpha P^{[[-B+C]]}(s) := \{\alpha P(s) \sim B\} \bigcup C$$

$$\tau P^{[[-B+C]]}(s) := \tau P(s), \text{ for } s \in trP^{[[-B+C]]}.$$

First we define **mutually recursive** process as follows:

$X = (X_1, \ldots, X_n) \in \prod^n$ is mutually recursive if for every i and trace $s \in trX_i$, the postprocess X_i/s has a representation

$$X_i/s = f(X_1, \ldots, X_n)$$

X is mutually recursive iff X is the solution of the recursive equation

$$Y = f(Y), Y \uparrow 0 = X \uparrow 0$$

where each component f_i of f has the form:

$$f_i(X) = (a_{i1} \rightarrow f_{i1}(X) | \ldots | a_{ik_i}(X))_{(A_i, \tau_i)}$$

$Y \in \prod$ is a **finitely recursive process (FRP)** if it can be represented as

$$X = f(X)$$

$$Y = g(X)$$

This is similar to the difference equation:

$$x(t+1) = f(x(t)), y(t) = g(x(t)), t = 0, 1, \ldots$$

is a finite representation of the process $y(t)$ (see [105]).

11 Performance evaluation via perturbation analysis

In this section, we review a framework for analyzing and evaluating the performance of discrete event dynamic systems (DEDS) called perturbation analysis (PA) [40, 93, 195]. The approach used in this framework is a quantitative approach that focuses on the performance measures of DEDS. There are other state space approaches that concentrate on the qualitative aspects of DEDS [129, 145, 159, 161], however, we shall concern ourselves only with the PA technique.

Discrete event dynamic systems (DEDS) are dynamic systems (typically asynchronous) in which state transitions are triggered by the occurrence of discrete events in the system. Many existing dynamic system have a DEDS structure, manufacturing systems and communication systems are just two of them. The PA approach to analyzing DEDS is different from the analysis techniques for the state space approach, the existence of a consistent and pre-defined automata-like model of the system under consideration is not necessary to perform PA. For example, if we consider a serial production line with M stations with a queue space of size K_i for each station. Then the total number of states for such a system would be $(\prod_{i=1}^{M}(K_i+1))(2^M)$, which can amount to billions for relatively small values of K_i and M. It is quite clear that modeling such systems as finite state machines is inefficient, if not impossible. It should also be mentioned that the finite state machines approach is more suitable for answering qualitative rather than quantitative questions.

Perturbation analysis (PA) is a technique that calculates the sensitivity of performance measures of DEDS with respect to system parameters by analyzing its sample path. The object of PA is to obtain the perturbed performance from a nominal experiment or sample path without doing a perturbed experiment. To avoid doing more than one experiment or simulate a perturbed experiment is the goal of PA.

11.1 Infinitesimal Perturbation Analysis (IPA)

To present the idea behind IPA, we shall first introduce a simple system (see Figure 8). It consists of a buffer, call it A, where messages arrive and are placed in a FIFO queue, and is connected via a link to another buffer, call it B, where the messages are received.

Consider the following definitions:

$$\begin{aligned} \theta &= \text{link service time (s/bit)} \\ H &= \text{header length (bits)} \\ L_i &= \text{length of message } i \text{ (bits)} \end{aligned}$$

We define the "service time" to be the time it takes to transmit a message i from A to B assuming the message does not wait in the queue before it gets sent. We denote this by

$$\begin{aligned} X_i &= (H + L_i)\theta \\ &= \gamma + L_i\theta. \end{aligned} \tag{1}$$

Let us also define the "system time", t_i, to be the time since a message i arrives at A till it is completely received by B. Finally let us call our performance measure $T(\theta, \gamma)$. This can be

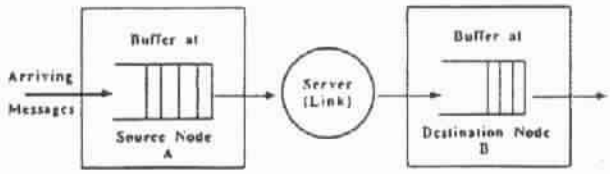


Figure 8: Link in a communication network.

approximated by using the mean system time, $\hat{T}(\theta, \gamma, N)$, where

$$\hat{T}(\theta, \gamma, N) = \left(\frac{1}{N}\right) \sum_{i=1}^{N} t_i. \tag{2}$$

Note that as $N \to \infty$, $\hat{T}(\theta, \gamma, N)$ converges to $T(\theta, \gamma)$.

For sensitivity estimates, we use $dT/d\theta$ and $dT/d\gamma$. A good estimate for $dT/d\theta$ is

$$\hat{F} = [\hat{T}(\theta + \Delta\theta, \gamma, N) - \hat{T}(\theta, \gamma, N)]/\Delta\theta. \tag{3}$$

Similarly a good estimate for $dT/d\gamma$ is

$$\hat{G} = [\hat{T}(\theta, \gamma + \Delta\gamma, N) - \hat{T}(\theta, \gamma, N)]/\Delta\gamma. \tag{4}$$

As can be seen, to obtain the estimates above one needs one more experiment at $\theta + \Delta\theta$ and another at$\gamma + \Delta\gamma$.

The problem here is to choose a value for $\Delta\theta$ (and similarly $\Delta\gamma$). For, if we choose to large a value we will not get a good estimate of the gradient. On the other hand, if we choose $\Delta\theta$ to be too small, we may amplify the noise interference present in $\hat{T}(\theta, \gamma + \Delta\gamma, N)$ and $\hat{T}(\theta, \gamma, N)$. In this section, however, we will not concern ourselves with this experimental problem.

11.1.1 An Unperturbed Experiment

Figure 9 displays the time evolution for a sequence of messages, that arrive and depart the buffer of A, within a certain period of time. Where A_i is the time between the arrival of M_{i-1} and M_i (with the exception that A_1 is from the start of the experiment). We define a busy period (BP) to be the time when the system is busy processing messages.

In our example, we start off with the buffer empty, and have to wait a time of length A_1 for the first message to arrive, and another X_1 for the message to be completely transmitted (hence total time is $A_1 + X_1$). However, during this time M_2, followed by M_3 arrive at the queue and have to wait for M_1 to get fully transmitted. In the case of M_2 the arrival time is $A_1 + A_2$ and the departure time is $A_1 + X_1 + X_2$. More generally, M_i has an arrival time of $t_0 + \sum_{j=2}^{i} A_j$ and a departure time of $t_0 + \sum_{j=1}^{i} X_j$, where $t_0 = A_1$. Hence we can define the system time to be

$$(t_0 + \sum_{j=1}^{i} X_j) - (t_0 + \sum_{j=2}^{i} A_i) = \sum_{j=1}^{i} X_j - \sum_{j=2}^{i} A_j. \tag{5}$$

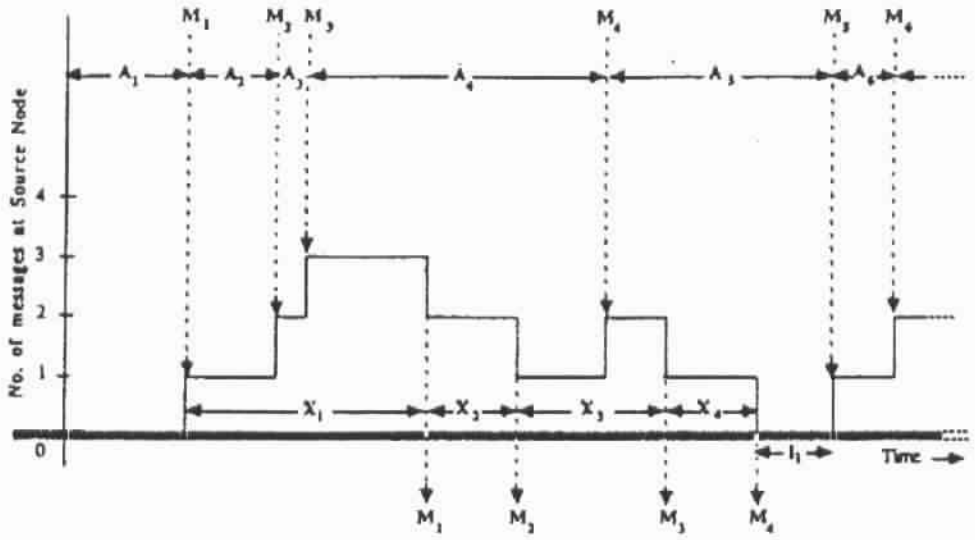


Figure 9: Time evolution of the experiment.

where the sum is zero for the case when $i = 1$. Note that this sum only holds up until the time of the complete departure of the fourth message (i.e. after the first busy period). Therefore, we can rewrite the system time (as would apply to our specific example) in the following way :

$$\sum_{i=1}^{4}\sum_{j=1}^{i} X_j - \sum_{i=1}^{4}\sum_{j=2}^{i} A_j. \tag{6}$$

or more generally, we can define it for the m^{th} busy period as follows :

$$\sum_{i=1}^{n_m}\sum_{j=1}^{i} X_{k_m+j} - \sum_{i=1}^{n_m}\sum_{j=2}^{i} A_{k_m+j}. \tag{7}$$

Hence the average system time of a message can be written as

$$\hat{T}(\theta,\gamma,N) = \left(\frac{1}{N}\right)\sum_{m=1}^{M}\sum_{i=1}^{n_m}\left(\sum_{j=1}^{i} X_{k_m+j} - \sum_{j=2}^{i} A_{k_m+j}\right). \tag{8}$$

11.1.2 Performing the IPA

We now consider the experiment at hand with the link service time set at $\theta + \Delta\theta$ (the *perturbed experiment*). In this case we will have an increase in the transmission time

$$\begin{aligned} \Delta X_i &= (H + L_i)\Delta\theta \\ &= (\Delta\theta/\theta)X_i \end{aligned} \tag{9}$$

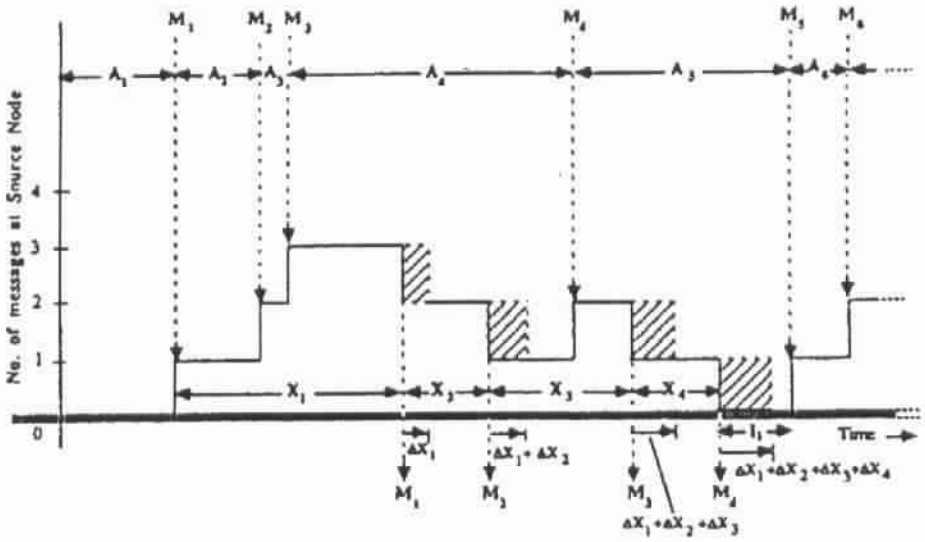


Figure 10: Perturbations in the sample path for case i.

This means that M_1 will take ΔX_1 longer to get fully transmitted, hence M_2 will take $\Delta X_1 + \Delta X_2$, and so on. Hence in the first busy period we have an increase in the system time

$$\begin{aligned} \Delta t_i &= \textstyle\sum_{j=1}^{i} \Delta X_j \\ &= (\Delta\theta/\theta) \textstyle\sum_{j=1}^{i} \Delta X_j \end{aligned} \tag{10}$$

However, when we move to the next busy period we must take into consideration two possibilities. Has the effect of $\Delta\theta$ caused M_4 to get completely transmitted after M_5 arrives? If this is not the case (see Figure 10) then the next busy period can be represented using equation (10). On the other hand, if this is the case then, returning to our example, we can see from Figure 11 that

$$\Delta t_5 = \Delta S_1 + \Delta X_5. \tag{11}$$

where ΔS_1 is the time where the first busy period has overlapped with the second. Hence, it follows that

$$\Delta t_6 = \Delta S_1 + \Delta X_5 + \Delta X_6. \tag{12}$$

in other words

$$\Delta t_i = \Delta S_1 + (\Delta\theta/\theta) \sum_{j=1}^{i} X_j. \tag{13}$$

We can generalize the equations further so as to represent the m^{th} busy period (let $\Delta S_{m-1}(\Delta\theta)$ be the amount BP_{m-1} overlaps with the arrival of M_{k_m+1}).

$$\Delta t_{k_m+i} = \Delta S_{m-1}(\Delta\theta) + (\Delta\theta/\theta) \sum_{j=1}^{i} X_{k_m+j}, \qquad i \leq n_m. \tag{14}$$

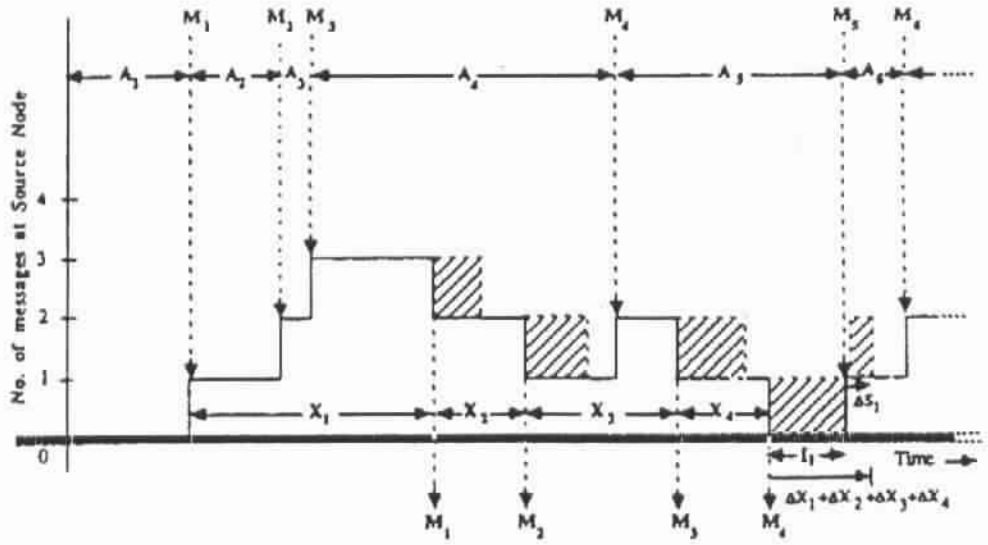


Figure 11: Perturbations in the sample path for case ii.

Note that $\Delta S_{m-1}(\Delta\theta)$ includes all effects of the previous busy periods. We are now ready to define the average system time after performing the perturbation:

$$\hat{T}(\theta,\gamma,N) = \left(\frac{1}{N}\right)\sum_{m=1}^{M}\sum_{i=1}^{n_m}[\Delta S_{m-1}(\Delta\theta) + (\Delta\theta/\theta)\sum_{j=1}^{i} X_{k_m+j}]. \tag{15}$$

We are now ready to define the sensitivity of $\hat{T}$ with respect to θ:

$$dT/d\theta = \lim_{\Delta\theta\to 0}\lim_{N\to\infty}\Delta\hat{T}(\theta,\gamma,N)/\Delta\theta. \tag{16}$$

Now we assume that as the number of messages increases the summations of busy periods' overlaps becomes negligible. In other words:

$$\lim_{\Delta\theta\to 0}\lim_{N\to\infty}\left(\frac{1}{N}\right)\sum_{m=1}^{M}\sum_{i=1}^{n_m}\Delta S_{m-1}(\Delta\theta) = 0. \tag{17}$$

Note that we will provide a reason for this assumption later on.
Hence, the correct measure of $dT/d\theta$ is reduced to

$$dT/d\theta = \lim_{N\to\infty}\left(\frac{1}{N}\right)\sum_{m=1}^{M} H_m/\theta. \tag{18}$$

where

$$H_m = \sum_{m=1}^{M}\sum_{i=1}^{n_m} X_{k_m+j}. \tag{19}$$

So finally the gradient estimate can be defined to be

$$\hat{g}_\theta(N) = (\sum_{m=1}^{M} H_m)/(N\theta). \tag{20}$$

We now try to estimate $dT/d\gamma$. We note that the equation

$$X_i = \gamma + L_i\theta. \tag{21}$$

tells us that γ is independent of L_i and θ.Therefore

$$\Delta X_i = \Delta\gamma.. \tag{22}$$

It follows that

$$\begin{aligned} \Delta t_i &= \textstyle\sum_{j=1}^{i} \Delta\gamma \\ &= \Delta\gamma \textstyle\sum_{j=1}^{i} 1 \end{aligned} \tag{23}$$

Hence our estimator is trivially

$$\hat{g}_\gamma(N) = (\sum_{m=1}^{M} \sum_{i=1}^{n_m} \sum_{m=1}^{M} 1)/N. \tag{24}$$

Hence we can implement the following algorithm to calculate both $dT/d\gamma$ and $dT/d\theta$ at the same time.

1. Initialize: Set J, $XSUM$, $JSUM$, $HSUM$, $GSUM = 0$;
   Set $THETA = \theta$;
2. Update: At departure of next message (with service time observed to be XJ);
   1.1) $J + 1$
   1.2a) $XSUM = XSUM + XJ$
   1.2b) $JSUM = JSUM + 1$
   1.3a) $HSUM = HSUM + XSUM$
   1.3b) $GSUM = GSUM + JSUM$
   1.4) If link is now idle then $XSUM = 0$ and $XSUM = 0$
3. Test: If $J = N$ then go to OUTPUT else goto UPDATE ;
4. Output: $dT/d\theta \approx HSUM/(N * THETA)$;
   $dT/d\gamma \approx GSUM/N$;

It was shown that under the assumptions of small perturbation values and in the near-absence of "dramatic" changes in the system's behavior due to the perturbation (i.e. assuming very little overlap between the busy periods, or, in other words, the system has the property that $\lim_{\Delta\theta\to 0} \lim_{N\to\infty} \left(\frac{1}{N}\right) \sum_{m=1}^{M} \sum_{i=1}^{n_m} \Delta S_{m-1}(\Delta\theta) = 0$) that an experimental estimate, which converges to the true value of $dT/d\theta$ as $N \to \infty$, can be easily computed while the nominal (unperturbed) experiment is evolving . It should be noted that this gradient estimate is an infinitismal

PA (IPA) estimate, and for "sufficiently small" $\Delta\theta$ the IPA estimate will be equal to the finite difference estimator. In other words we say

$$\hat{g}_\theta(N) = \Delta\hat{T}(\theta, \gamma, N)/\Delta\theta, \qquad \Delta\theta \leq \epsilon. \tag{25}$$

where ϵ is very small.

However, one should notice that the correct definition of the gradient involves letting $N \to \infty$ first and *then* $\Delta\theta \to 0$ for convergence to $dT/d\theta$, but as can be noticed in (25), the order in which we take limits is reversed, for we let $\Delta\theta \to 0$ then let $N \to \infty$. *In order to be able to switch the limits we must make the assumption that the system satisfies :*

$$\lim_{N\to\infty} \lim_{\Delta\theta\to 0} \frac{\Delta\hat{T}(N;\Delta\theta)}{\Delta\theta} = \lim_{\Delta\theta\to 0} \lim_{N\to\infty} \frac{\Delta\hat{T}(N;\Delta\theta)}{\Delta\theta} \tag{26}$$

For it is this assumption that make it feasible to do the estimation for very small $\Delta\theta$ and *then* find the estimator for large N (hence changing the order of taking limits).

Then it follows that

$$\lim_{N\to\infty} \hat{g}_\theta(N) = dT/d\theta. \tag{27}$$

For the class of systems where (26) holds, hence, we can make excellent use of the PA experiment.

11.2 IPA for a GI/G/1 System

We now consider the PA experiment when applied to a GI/G/1 queue. We start by defining two sets of i.i.d(independent and identically distributed) random variables. First we have the set of r.v.'s

$$\{A_1, A_2, ...\}. \tag{28}$$

this represents the sequence of interval times during a given experiment, and

$$\{X_1, X_2, ...\}. \tag{29}$$

represents a sequence of service times. Next, we assume that X_1 is dependent on θ. Finally, we make an assumption that the system is stable, that is $E(X_i) < E(A_i)$. We are interested in the mean service time $T(\theta)$. This - as mentioned earlier - is close to the value of $\hat{T}(\theta, N)$ for large N,or

$$\lim_{N\to\infty} \hat{T}(\theta, N) = T(\theta). \tag{30}$$

To estimate $dT/d\theta$, we first make the assumption that the r.v.'s $X_i(\theta)$ are uniformly differentiable. We make use of this assumption and of (9) and rewrite the equation (14) as

$$\Delta t_{k_m+i} = \Delta S_{m-1}(\Delta\theta) + \sum_{j=1}^{i} \Delta X_{k_m+j}. \tag{31}$$

Also, we have

$$dX_i/d\theta = \lim_{\Delta\theta\to 0} \Delta X_i/\Delta\theta. \tag{32}$$

Hence, as before we try and estimate the sensitivity. We have

$$\begin{aligned} dT/d\theta &= \lim_{N\to\infty} \lim_{\Delta\theta\to 0} \left(\frac{1}{N}\right) \sum_{m=1}^{M} \sum_{m=1}^{M} \sum_{i=1}^{n_m} \Delta X_{k_m+j}/\Delta\theta \\ &= \lim_{N\to\infty} \left(\frac{1}{N}\right) \sum_{m=1}^{M} \sum_{m=1}^{M} \sum_{i=1}^{n_m} dX_{k_m+j}/d\theta \end{aligned} \tag{33}$$

Thus our IPA estimator is finally

$$\hat{g}(N) = \left(\frac{1}{N}\right) \sum_{m=1}^{M} \sum_{i=1}^{n_m} \sum_{j=1}^{i} dX_{k_m+j}/d\theta. \tag{34}$$

11.2.1 Sensitivity Analysis for Random Parameters

Earlier in our development, we stated that X_i is dependent on θ. We now need to elaborate more on this matter in order to display some features of the PA experiment. X_i can be dependent on θ in one of two cases. In the first case

$$X_i = (H + L_i)\theta. \tag{35}$$

Therefore

$$\begin{aligned} dX_i/d\theta &= (H + L_i) \\ &= X_i/\theta. \end{aligned} \tag{36}$$

However, there are other systems where

$$X_i = H + L_i + \theta. \tag{37}$$

Then, trivially

$$dX_i/d\theta = 1. \tag{38}$$

What can be observed from the two results above is that $\Delta\theta$ *does not appear on the RHS. This is the whole idea behind the IPA, for it means that we can find the estimate without having to repeat the experiment at* $\Delta\theta$*! Furthermore, in the former result, we need not even concern ourselves with the distribution of the r.v.* X_i*. In the latter, case we don't even need to know* θ*.*

We can now safely make the assumption that $dX_i/d\theta$ can be expressed as $\psi(X_i, \theta)$.

The following is an algorithm for estimating $dX_i/d\theta$:

1. Initialize: Set J, $XSUM$, $HSUM = 0$;
2. Update: At departure of next message (with service time observed to be XJ);
   1.1) $J + 1$;
   1.2) $XSUM = XSUM + PSI(XJ, THETA)$;
   1.3) $HSUM = HSUM + XSUM$;
   1.4) If link is now idle then $XSUM = 0$;
3. Test: If $J = N$ then go to OUTPUT else goto UPDATE ;
4. Output: $dT/d\theta \approx HSUM/N$;

11.2.2 Consistency of IPA

We now want to insure that the assumption that

$$\lim_{N\to\infty} \hat{g}_\theta(N) = dT/d\theta. \tag{39}$$

is solid. But, assuming for the moment that the above assumption is true, we can also make the following inference :

$$\lim_{N\to\infty} E(\hat{g}_\theta(N)) = dT/d\theta. \tag{40}$$

We can prove this fact for an M/M/1 (due to the simplicity of the proof). This system is described by an exponentially distributed arrival times, with rate λ and mean $1/\lambda$, and by an exponentially distributed service times with mean θ. Finally the traffic intensity is defined by $\rho = \lambda\theta$. We are also given

$$\begin{aligned} T(\theta) &= \theta/(1-\rho) \\ E(B) &= \theta/(1-\rho) \\ E(B)^2 &= 2\theta^2/(1-\rho)^3 \end{aligned} \tag{41}$$

where B is a r.v. for the time length of an arbitrary busy period. Differentiating T, we get

$$dT/d\theta = 1/(1-\rho)^2. \tag{42}$$

Also since we can see that θ is a scale parameter of X_i, we have

$$dX_i/d\theta = X_i/\theta. \tag{43}$$

Since we are assuming that the estimate is consistent we can say

$$\begin{aligned} g &= \textstyle\sum_{i=1}^{j} dX_i/d\theta \\ &= (1/\theta)\textstyle\sum_{i=1}^{j} X_i \end{aligned} . \tag{44}$$

Looking at $\sum_{i=1}^{j} X_i$ we can see that it is the time from the start of a busy period till the departure of the j^{th} message in this busy period. This summation can be rewritten as the time from the start of the busy period to the time of the arrival of message j (denoted by z_j), plus the system time of the message. Or,

$$g = (z_j + t_j)/\theta. \tag{45}$$

Now working with the expected value of g (to simplify our proof) we get

$$E(g) = (E(z_j) + E(t_j))/\theta \tag{46}$$

Analyzing the above equation we see that the expected system time was defined by us earlier to be $T(\theta)$. On the other hand, $E(z_j)$ is the expected time for the message to arrive. Hence, one of the following two cases may be the situation. Either the server is idle (denote that by I), or the system is busy (denote that by b). In other words

$$E(z_j) = (E(z_j|I)p_I + E(z_j|b)p_b. \tag{47}$$

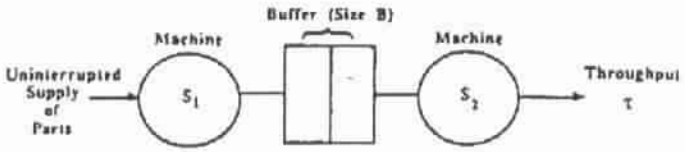


Figure 12: A Simple Production Line.

But when the system is idle there is no busy period, therefore z_j is zero. Therefore

$$E(z_j) = E(z_j|b)p_b. \tag{48}$$

where p_b is the utilization of the server ρ, and $E(z_j|b)$ is the average time of a busy period seen by a random arrival into the BP (which has been found to be $E(B)^2/2E(B)$). Thus

$$E(z_j) = \rho E(B)^2/2E(B). \tag{49}$$

going back to $E(g)$, we now have

$$E(g) = (\rho E(B)^2/2E(B) + T(\theta))/\theta. \tag{50}$$

Substituting the values the we are given in (41) we get

$$\begin{aligned} E(g) &= (\rho\theta/(1-\rho)^2 + \theta/(1-\rho))/\theta \\ &= 1/(1-\rho)^2. \end{aligned} \tag{51}$$

thus proving the assumption made in (39).

11.3 IPA for General Networks

In the previous section, the main ideas of infinitismal perturbation analysis were illustrated using a single server queue model of a communication link. To make use of IPA in realistic situations, we have to look at IPA for more general systems. We are going to address the problem of finding IPA algorithms for the case of a simple production line with just two machines and then for a general network of servers.

11.3.1 IPA for a Simple Production Line

IPA can be performed for a simple production line consisting of two servers (machines) and a buffer in between as shown in Figure 12. The production line can be thought of as a system consisting of two computers and one buffer.

Server 1 (S_1) is a machine whose cycle time depends on a parameter θ_1. We can assume that S_1 has an uninterrupted supply of parts to work on. After S_1 finishes its work cycle on a part, it places the part in the buffer. The second machine S_2 picks one part from the buffer, works on it for a cycle time (which depends on a parameter θ_2) and then releases it to a finished goods area.

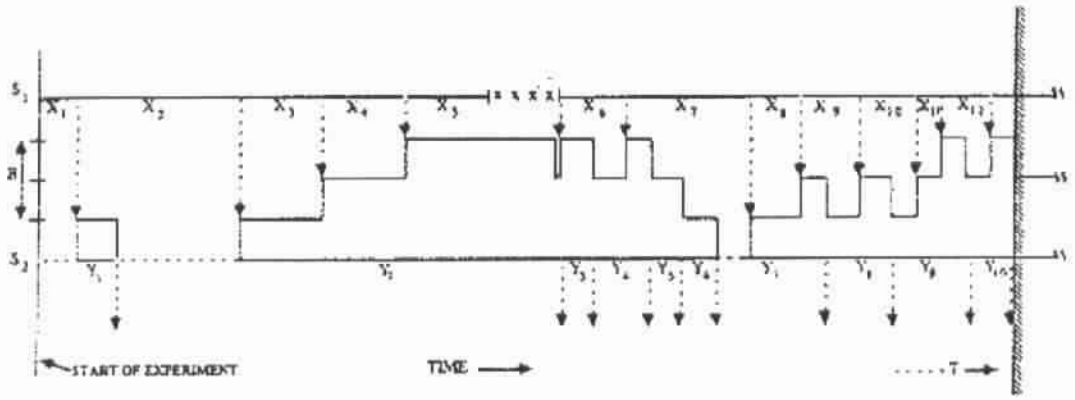


Figure 13: Nominal sample path for the production line.

The size of the buffer is B. If the buffer is full when S_1 completes a part then the part stays at S_1, which is then unable to work on another part and is said to be *blocked*. S_1 remains blocked until S_2 finishes its current cycle, releases its part, and takes the next part from the buffer, thereby releasing a buffer space. We shall assume that all transfers take place in a negligible amount of time, and that the finished goods area is never blocked. The performance measure we shall consider of interest for this system is its steady state throughput (number of parts produced per unit time) which we shall denote $\tau(\theta_1, \theta_2)$. We can define an experiment on this system, starting with no parts in S_1, S_2, or the buffer, and ending when the Nth part is completed by S_2. If T is the length of time for this experiment, then the experimental estimate of the throughput is

$$\hat{\tau}(\theta_1, \theta_2, N) = N/T \tag{52}$$

Under some conditions, this estimate will satisfy

$$\lim_{N \to \infty} \hat{\tau}(\theta_1, \theta_2, N) = \tau(\theta_1, \theta_2) \tag{53}$$

which is desired for a good experimental estimate.

A typical sample path is shown in Figure 13 with $N = 10$, X_i and Y_i denotes the cycle time for S_1 and S_2 for the ith part. The vertical axis represents the number of parts at S_2 and at the buffer. The size of the buffer B is 2 for this example, part i is denoted by P_i and dashed lines implies that S_2 is idle, crosses implies that S_1 is blocked. Our goal then is to develop an IPA algorithm to estimate $d\tau/d\theta$ for this system. Introducing a perturbation $\Delta\theta$ in this system, the perturbed sample path is shown in Figure 14. Where ΔX_i $(X_i(\theta_1 + \Delta\theta_1) - X_i(\theta_1))$ denotes the change in cycle times at S_1 due to a change $\Delta\theta_1$ in the parameter θ_1. It should be clear that there is an implicit assumption for the perturbed path shown in Figure 14, namely that the perturbations are small enough so that the order of events does not change, such assumption is standard in IPA.

With the above assumption, stating the IPA algorithm becomes particularly simple. Letting AC_1 and AC_2 be accumulators associated with S_1 and S_2, AC_j is the perturbation at S_j for the

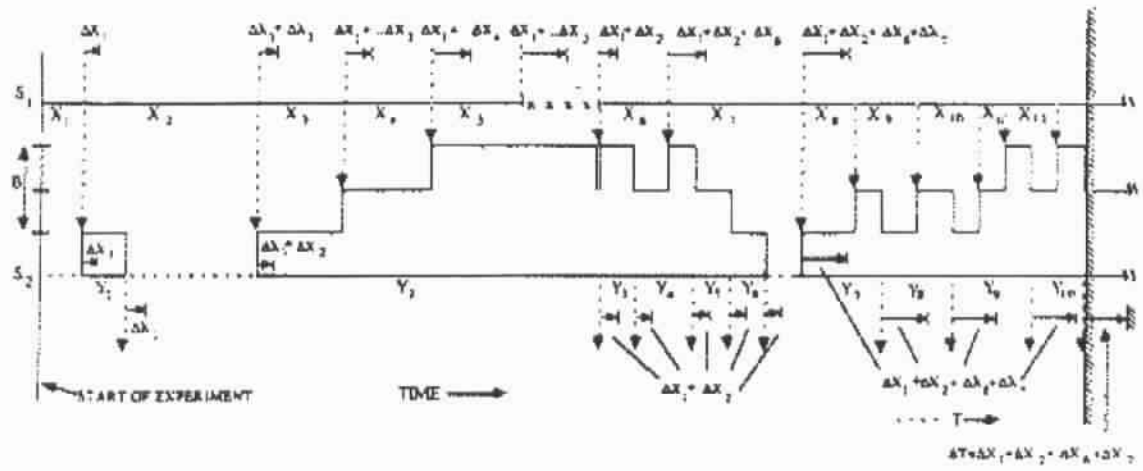


Figure 14: Perturbations in the sample path for the production line.

last part that left S_j, and the arrows ($\rightarrow$ |) shows the values of the accumulators. Then we can develop three rules, the first is that whenever a part P_i has been served at S_1 the first accumulator is incremented by ΔX_i, the second is that if P_i finds S_2 idle, then AC_2 gets the value of AC_1 and finally if P_k unblocks S_1 by departing from S_2 then AC_1 gets the value of AC_2. We can then proceed to write the algorithm for calculating the gradient of θ_1.

At the end of the experiment, $\Delta T = AC_2$, and as shown above AC_2 is the sum of some of the ΔX_i values, say for $i \in l$. Under the assumption that the random values $X_i(\theta)$ have the property that $dX_i/d\theta$ can be expressed as $\psi(X_i, \theta)$, we can say that

$$\frac{dT}{d\theta_1} = \lim_{\Delta\theta_1 \to 0} \frac{AC_2}{\Delta\theta_1} = \sum_{i \in l} \frac{dX_i}{d\theta_1} = \sum_{i \in l} \psi(X_i, \theta_1) \tag{54}$$

and since N is fixed by definition of the experiment, then

$$\frac{d\hat{\tau}}{d\theta_1} = -(N/T^2)\frac{dT}{d\theta_1} = -(N/T^2)\sum_{i \in l} \psi(X_i, \theta_1) \tag{55}$$

Which implies that if we accumulate $\psi(X_i, \theta)$ instead of ΔX_i, in the first rule above, and call the accumulators A_1 and A_2, then after the experiment is performed, the value $-(N/T^2)A_2$ will be the IPA estimate of $d\hat{\tau}/d\theta_1$. The algorithm is then developed as follows :

1. Initialize: Set A_1, $A_2 = 0$;
   Set $THETA1 = \theta_1$;
2. Update: Whenever a part (say P_i) completes service, check these conditions :
   1) If P_i completed service at S_i then
      $A_1 \leftarrow A_1 + PSI(X_i, THETA1)$;
   2) If P_i leaves S_1 and terminates an idle
      period of S_2 then $A_2 \leftarrow A_1$;
   3) If P_i leaves S_2 and terminates a blocked

period of S_1 then $A_1 \leftarrow A_2$;

3. Test: If S_2 has completed N parts go to OUTPUT
else goto UPDATE ;

4. Output: Let T be the total time since the start of the experiment;
The IPA estimate of $d\tau/d\theta$ is $-(N/T^2)A_2$.

11.3.2 IPA for General Networks with Finite Buffers

Considering a general network with finite buffers, having a single server at each station, we can generalize the algorithm described above easily to allow for more than two servers. It should be noted that the only times when perturbations propagate from one server to another are when idle or blocked intervals are terminated by a customer moving from one server to another. Thus the propagation rules 2 and 3 in the above algorithm can be modified by allowing for any servers S_i and S_k instead of S_1 and S_2 and naming the associated accumulators A_i and A_k and thus replacing $A_2 \leftarrow A_1$ by $A_k \leftarrow A_i$. In general network it is possible to have a situation of "chain" blocking, where, for example, S_k is blocked by S_i, and then in turn the buffer at S_K gets full and it ends up blocking S_j. In this case we just need to implement the propagation for each unblocked server in turn, but there is no change in the rule. A further generalization would be to change the first condition statement in the 2-server algorithm to allow the use of the accumulators associated with different servers. It is also possible to state the algorithm in such a way so that it can compute all K gradients at the same time as follows :

(A_{ij} is the accumulator at S_i for gradient with respect to θ_j)

1. Initialize: Set A_{ij}, $i = 1, ..., K$; $j = 1, ..., K$;
Set $THETA_i = \theta_i$, $i = 1, ..., K$;

2. Update: Whenever a customer (say C) completes service, check these conditions :

1) If C completed service at S_i then
$A_{ii} \leftarrow A_{ii} + PSI(i, X, THETA_i)$;

2) If C leaves S_1 and terminates an idle
period of S_m then $A_{mj} \leftarrow A_{ij}$,
for $j = 1, ..., K$; (If there is a chain of blocking
then continue this procedure through the chain)

3. Test: If S_{end} has completed N parts go to OUTPUT
else goto UPDATE ;

4. Output: Let T be the total time since the start of the experiment;
The IPA estimates of the K gradients $d\tau/d\theta_j$
$(j = 1, ..., K)$ are $-(N/T^2)A_{endj}$ $(j = 1, ..., K)$.

11.4 Extensions of IPA

In some cases, the IPA technique discussed aboye will fail to work. One instance might be due to the assumption that small changes in the system parameter θ will not cause coalescing of busy periods in a GI/G/1 queue because of small $\Delta\theta$. Suppose that the performance measure of interest is the average number of messages sent between idle periods of a communication link. If we model the link as a single server queue, this performance measure is the average number of customers served in a busy period (BP). Denoting this average by $\beta(\theta)$, then a simple experimental estimate for $\beta(\theta)$ would be to observe M BPs and then let

$$\hat{\beta}(\theta, M) = \left(\frac{1}{M}\right) \sum_{m=1}^{M} n_m \tag{56}$$

where n_m is the number of customers served in BP_m. Considering the arguments presented in the IPA, we can see that IPA is based entirely on the assumption that no BPs will coalesce. If we make $\Delta\theta$ small enough so that no BPs coalesce, then each n_m value will remain the same, so that there will be no change in the estimate of the performance measure. Thus, the IPA estimate of sensitivity will be zero ! It is clear that this is wrong and thus IPA failed in this example. IPA ignores the effects of some events in the system, when the probability of occurrence of these events, multiplied by the effect of the events on the performance is significant, IPA fails. This motivates some extensions which enable gradient estimation for a wider class of systems.

11.4.1 Smoothed Perturbation Analysis (SPA)

Motivated by the failure of IPA to work for the simple case above, the idea of using conditional probabilities was introduced to develop an extension for the IPA. A conditioning variable can be used to decompose the gradient estimate expectation expression. The fact that more information is used in developing the conditional probability counts for the "smoother" kind of performance measure estimate curve that is obtainable by using this method. For example, we can ask the question, for a given $\Delta\theta$, what is the *expected* change in the value of n_i, based on the observed BP_i.

11.4.2 Extended Perturbation Analysis (EPA)

For systems that can be represented by Markov chains, a new approach that may overcome the potential inconsistency of IPA can be applied. The idea behind the extended perturbation analysis is the fact that the perturbed and unperturbed systems should be statistically evolving similarly once they enter a common state x, due to their Markovian property. This method works by choosing a finite $\Delta\theta$ and predicting, from the nominal path, where the perturbed path would have branched to a different state, say y, while the nominal path continues in, say, state x. Up to this point, an IPA-like estimator is used to compute the effects of perturbation, but at this point, the computation is "frozen". The algorithm then waits for the system to enter state y during the nominal path, then EPA restarts. When an event order change occurs, the state sequences of the nominal path (NP) and the perturbed path (PP) may or may not start to differ depending

Figure 15: Markovian state space sequences.

on whether some discontinuous change is involved (e.g., a job originally going to server A may now go to server B). As shown in Figure 15, if ω_1 and ω_2 are two state sequences of a Markov DEDS and the state sequence jumps on from S_s on ω_1 to S_p on ω_2 instead of S_h on ω_1, subsequent perturbations involving state changes may cause further deviations so that a perturbed path could be made up from segments of state sequences from ω_1, ω_2,...,ω_j,...

We can see right way that EPA cannot be as efficient as IPA, since it may remain "inactive" for significant sections of the nominal experiment. However, there are two factors that make its performance better than one might expect. The first is that in most applications we do the gradient estimation with respect to a number of parameters simultaneously, it will probably turn out that several of the gradient computations are "active", on average, during the observations and the savings is still better compared to multiple experimentation. The second is that from a practical point of view, one can often aggregate the states of the system to fewer subsets, and use the aggregate state to decide whether to activate or deactivate the EPA calculation. Not only does this keep the computation active for longer segments of the experiment, but it also enables EPA to be applied to non-Markovian systems.

11.4.3 Other Perturbation Techniques

Another Perturbation technique is finite perturbation analysis (FPA), this technique was introduced to overcome the IPA assumption that events do not change order. However, FPA considers changes in order of events to a pre-specified limit, for example, it may consider only "first order" changes, that is, changes in the order of adjacent events, and ignores any effects of changes in order beyond adjacent events. The way it works then is to introduce perturbations and propagate them while observing the nominal path, but limiting its calculations by only extrapolating to

predict the effect of such changes in order. Originally FPA was heuristic and experimental in nature, however, recent research has been performed to provide more theoretical foundations for it.

Other techniques to make IPA work include changing the system parameter under consideration to transform problems into "easier" versions, or to versions that have already been solved. Using a different representation for the system sometime helps in performing IPA.

11.5 Research Issues and Future Work

Many problems regarding discrete event dynamic systems in general, and perturbation analysis as an evaluation technique remains open. For example, performing PA for a discrete parameter θ is one such interesting problem. In practical systems, many parameters (such as buffer sizes, or number of servers at a station) are discrete in nature. It should be noticed that IPA, by its nature can be applied only to continuous parameters. Understanding and expanding the domain of IPA needs to be addressed, in fact, to "automate" the process of generating algorithms to calculate the sensitivity of a performance measure remains an open problem. To be able to construct a preprocessing stage, where its inputs are the system specification and the performance measure and parameters of interest, and the output as an IPA algorithm to be run while the nominal experiment is performed, is one challenging problem for researchers. More work still remains to be done on developing efficiency and accuracy measures for the PA output. Trying to get the "maximum" amount of information from a sample path is another long-term goal.

12 State Space DEDS Representation

In this section, we review a framework for analyzing and controlling discrete event dynamic systems (DEDS) [145]. The approach used in this framework is a state space approach that focuses on the qualitative aspects of DEDS. It considers the issues of stability, observability, stabilizability by output feedback and invertibility within this framework.

12.1 What is a discrete event dynamic system ?

Discrete event dynamic systems (DEDS) are dynamic systems (typically asynchronous) in which state transitions are triggered by the occurrence of discrete events in the system. Many existing dynamic system have a DEDS structure, manufacturing systems and communication systems are just two of them. The state space approach in representing and analyzing such systems will probably lead to more applications that might be incorporated into the framework of DEDS. It will be assumed in the development of the state space approach of analyzing DEDS that some of the events in the system are *controllable*, i.e, can be enabled or disabled. The goal of controlling DEDS is to "guide" the behavior of the system in a way that we consider "desirable". It is further assumed that we are able to observe only a subset of the event, i.e, we can only *see* some of the events that are occurring in the system and not all. In some cases we will be forced to

make decisions regarding the state of the system and how to control a DEDS based upon our observations only.

Next, we will discuss the finite state model of a DEDS. This model will be a simple nondeterministic finite-space automaton. Graphical representations for DEDS automatons will be used as examples to explain the definitions and ideas presented in the next four subsections.

12.2 Modeling

The discrete event dynamic systems under consideration can always be modeled by a nondeterministic finite-state automata with partially observable and controllable events. In particular, one can make the distinction between classical automata theory [98, 164, 114, 128] and the new representation of DEDS in terms of the state transitions. In classical automata the events are inputs to the system, whereas in DEDS the events are assumed to be generated internally by the system and the inputs to the system are the control signals that can enable or disable *some* of these events. We can represent our DEDS as the following quadruple :

$$G = (X, \Sigma, U, \Gamma)$$

where X is the finite set of states, Σ is the finite set of possible events, U is the set of admissible control inputs consisting of a specified collection of subsets of Σ, corresponding to the choices of sets of controllable events that can be enabled and $\Gamma \subseteq \Sigma$ is the set of observable events. Some functions can also be defined on our DEDS as follows :

- $d : X \rightarrow 2^{\Sigma}$
- $e : X \rightarrow 2^{\Sigma}$
- $f : X \times \Sigma \rightarrow 2^{X}$

where d is a set-valued function that specifies the set of possible events defined at each state, e is a set-valued function that specifies the set of events that cannot be disabled at each state, and f is the set-valued function that specifies state transitions from a state under different events. An output process can be formalized simply : whenever an event in Γ happens we see it, otherwise we don't see anything.

We can visualize the concept of DEDS by an example as in Figure 16 the graphical representation is quite similar to a classical finite automaton. Here, circles denote states, and events are represented by arcs. The first symbol in each arc label denotes the event, while the symbol following "/" denotes the corresponding output (if the event is observable). Finally, we mark the controllable events by ":u". Thus, in this example, $X = \{0, 1, 2, 3\}$, $\Sigma = \{\alpha, \beta, \delta\}$, $\Gamma = \{\alpha, \delta\}$, and δ is controllable at state 3 but not at state 1. Also $d(1) = e(1) = \{\alpha, \delta\}$, $d(3) = \{\delta\}$, $e(3) = \phi$, $f(0, \beta) = \{0, 3\}$ etc. A transition, $x \rightarrow^{\sigma} y$, consists of a source state, $x \in X$, an event, $\sigma \in d(x)$, and a destination state, $y \in f(x, \sigma)$.

In general, a DEDS automaton A is a nondeterministic finite state automaton, however, if $f(x, \sigma)$ is single valued for each $x \in X$ then A can be termed as a deterministic finite state

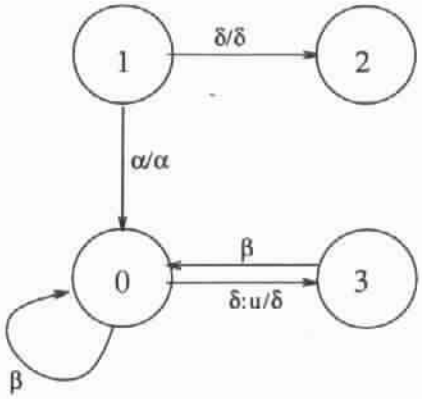


Figure 16: A Simple Example for DEDS.

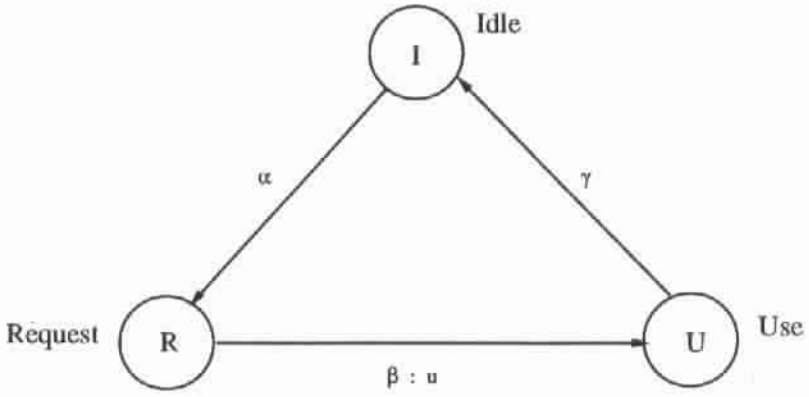


Figure 17: A Resource User Example.

automaton. A finite string of states, $\mathbf{x} = x_0 x_1 ... x_j$ is termed a path or a state trajectory from x_0 if $x_{i+1} \in f(x_i, d(x_i))$ for *all* $i = 0...j-1$. Similarly, a finite string of events $s = \sigma_1 \sigma_2 ... \sigma_j$ is termed an event trajectory from $x \in X$ if $\sigma_1 \in d(x)$ and $\sigma_{i+1} \in d(f(x, \sigma_1 \sigma_2 ... \sigma i))$ for *all* i, where we extend f to Σ^* via

$$f(x, \sigma_1 \sigma_2 ... \sigma_i) = f(f(x, \sigma_1 \sigma_2 ... \sigma_{i-1}), \sigma_i)$$

with $f(x, \epsilon) = x$. In our graphical example (Figure 16), $\alpha\beta\beta\delta$ is an event trajectory.

Another realistic and simple example for a DEDS can be modeled as a resource user (Figure 17). Where the Automaton will be a deterministic one in this case, with three states I (IDLE), R (REQUEST) and U (USE), and with transitions as shown.

Here we take $X = \{I, R, U\}$, $\Sigma = \{\alpha, \beta, \gamma\}$, $\Gamma = \{\alpha, \beta, \gamma\}$. The (two) control patterns corresponds to enabling and disabling event β at state R. A transition $R \rightarrow U$ may occur only when β is enabled. More interesting examples for using resources arise with the concurrent control of several of the above resource user example.

12.2.1 Generated Languages

A collection of strings $L \subset \Sigma^*$ is termed a language over the alphabet Σ [161]. For example, for any $x \in X$, $L(A, x)$ is a language over Σ which we refer to as the language generated by x in A. In our first example (Figure 16), $L(A, 0)$ can be expressed as $(\beta + \beta\delta)^*$, where "+" denotes the union of β and $\beta\delta$.

A language is termed a regular language if it can be expressed by using concatenations, unions and *. Since we use a nondeterministic finite automaton to represent a DEDS, and we know from classical automata theory that any nondeterministic finite automaton can be converted to a deterministic finite one, it will always be the case that the languages generated by a state in A are regular, as deterministic finite automata always produce regular languages as opposed to more powerful models such as pushdown automata, grammars and Turing machines. It will never be the case that a state will generate a palindrome language or a language like $\{\alpha^i\beta^i | \alpha, \beta \in \Sigma, i \in N\}$, where N is the set of natural numbers. A recognizer can always be constructed for such a regular language, it is also a fact that there exists a recognizer with the least possible number of states. Such a recognizer is termed minimal.

12.2.2 Ranges and Liveness

If we denote a transition labeled by σ by $\rightarrow^\sigma$, then we can similarly let $\rightarrow^s$ denote a string of transitions s and $\rightarrow^*$ denote any number of transitions, including no transitions. We can define the range of a state x by

$$R(A, x) = \{y \in X | x \rightarrow^* y\}$$

indicating the set of states that can reached from x, we can also define the range of a subset of states Q in X by

$$R(A, Q) = \bigcup_{x \in Q} R(A, x)$$

An algorithm for computing $R(A, X_0)$ for any $X_0 \subset X$ that runs in $O(n)$ where $n = |X|$ can be easily formalized as follows :

Let $R_0 = Q_0 = X_0$ and iterate

$$R_{k+1} = R_k \cup f(Q_k, \Sigma)$$
$$Q_{k+1} = R_{k+1} \cap \overline{R_k}$$

Terminate when $R_{k+1} = R_k$. Then, $R(A, X_0) = R_k$.

A state $x \in X$ is *alive* if $d(y) \neq \phi$ for *all* $y \in R(A, x)$. A subset Y of X is termed a *live set* if all $x \in Y$ are alive. A system A is termed *alive* if X is a *live set*.

12.3 Stability

In this section we discuss the notions of stability and the possibility of stabilizing a discrete event dynamic system. In particular, we are going to concentrate on stability notions with respect to the *states* of a DEDS automaton. Assuming that we have identified the set of "good" states, E,

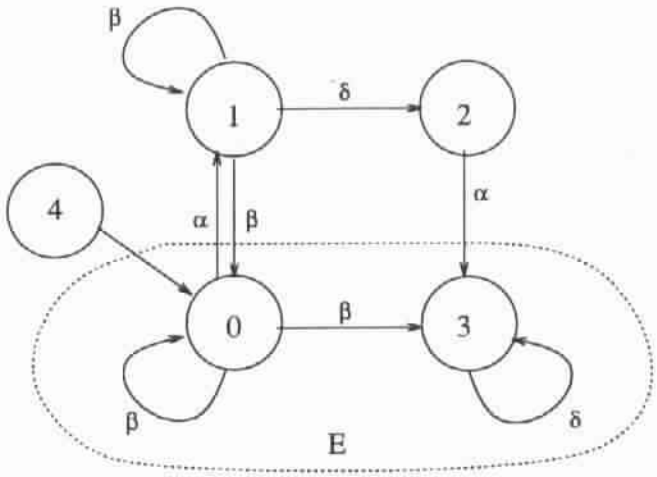


Figure 18: Stability Example.

that we would like our DEDS to "stay within" or do not stay outside for an infinite time, the problem would reduce to :

- Checking out whether all trajectories from the other states will visit E infinitely often.
- Trying to "guide" the system using the controllable events in a way such that the system will visit the "good" states infinitely often.

We shall start by defining and testing for different notions of stability and then discuss ways to stabilize a system. We shall start by assuming that the DEDS model under consideration is an uncontrolled system with perfect knowledge of the state and event trajectories ($\Sigma \cap \overline{\Gamma} = \phi$), to simplify developing the definitions and examples.

12.3.1 Pre-Stability

To capture the idea of stability , we can suppose that we have already identified a subset of states E in X that returning to E implies being in a position to continue desired behavior from that point on. We can define the notion of a state in the DEDS being stable with respect to E in two stages. The first stage will be the weaker notion and will be termed pre-stability. We say that $x \in X$ is pre-stable if *all* paths from x can go to E in a finite number of transitions, i.e, no path from x ends up in a cycle that does not go through E.

In Figure 18, states 0, 2, 3, and 4 are pre-stable, since all transitions from them can goto {0, 3} in a finite number of transitions. State 1 is not pre-stable since it will stay forever outside E if an infinitely long string of δ's occurs.

A definition of pre-stability can be formalized as follows :

Given a live system A and some $E \subset X$, a state $x \in X$ is pre-stable with respect to E (or E-pre-stable) if for all $\mathbf{x} \in \mathcal{X}(A, x)$ such that $|\mathbf{x}| \geq n$, there exists $y \in \mathbf{x}$ such that $y \in E$. We say that a set of states is E-pre-stable if all its elements are E-pre-stable and a system A is pre-stable if X is E-pre-stable.

The restriction for liveness can be flexible in the sense that if all the dead states are within E, then an automaton might still be E-pre-stable. It follows from the above definition that a state $x \in X$ is E-pre-stable iff $x \in E$ or $f(x, d(x))$ is E-pre-stable. The following algorithm computes the maximal E-pre-stable set X_p within a system :

Let $X_0 = E$ and iterate :

$$X_{k+1} = \{x | f(x, d(x)) \subset X_k\} \cup X_k$$

Terminate when $X_{k+1} = X_k$, then $X_p = X_k$.

In Figure 18, it can be noticed that $X_1 = X_2 = X_p = \{0, 2, 3, 4\}$.

12.3.2 Stability

The stronger notion of stability corresponds to returning to the set of "good" states E in a finite number of transitions following any excursion outside of E. Thus, given E, we define a state $x \in X$ to be E-stable if all paths go through E in a finite number of transitions and then visit E infinitely often.

As an example, in Figure 18, where $E = \{0, 3\}$, only 2 and 3 are stable states. State 1 is not stable since the system can loop at 1 infinitely. State 0 although in E is not stable since the system can make a transition to 1 and then stays there forever, the same applies to state 4.

We can use the previously defined notion of pre-stability and define a state to be E-stable if all the states in its reach are E-pre-stable. In Figure 18, 0 and 4 are not E-stable since they can reach 1, which is not E-pre-stable. We can define stability as follows :

Given a live A and $x \in X$, x is E-stable iff $R(A, x)$ is E-pre-stable. A $Q \subset X$ is stable if all $x \in Q$ are stable. A system A is stable if X is a stable set, from which we can conjecture that A is E-stable iff it is also E-pre-stable.

12.3.3 f-Invariance

A much stronger notion of stability can be described as "staying" within a given set of states. We thus define f-invariance for a subset Q in X as follows :

A subset Q of X is f-invariant if $f(Q, d) \subset Q$ where

$$f(Q, d) = \bigcup_{x \in Q} f(x, d(x))$$

It follows that any trajectory that starts in an f-invariant set stays in that set *forever*, it also follows that a set Q is f-invariant iff $R(A, Q) \subset Q$.

12.3.4 Pre-Stabilizability

In this section we introduce control and reconsider the stability notions discussed before. We try to "guide" our system or some states of it to behave in a way that we consider desirable.

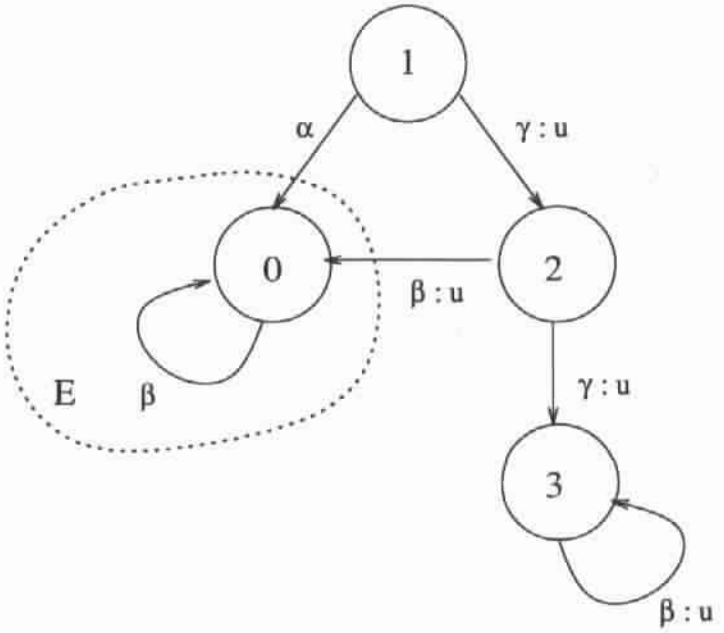


Figure 19: Example for the notion of Pre-Stabilizability.

Pre-stabilizability is described as finding a state feedback such that the closed loop system is pre-stable. We can then define pre-stabilizability formally as follows :

Given a live system A and some $E \subset X$, $x \in X$ is pre-stabilizable with respect to E (or E-pre-stabilizable) if there exists a state feedback K such that x is alive and E-pre-stable in A_K. A set of states, Q, is a pre-stabilizable set if there exists a feedback law $K(s)$ (A control pattern) so that every $x \in Q$ is alive and pre-stable in A_K, and A is a pre-stabilizable system if X is a pre-stabilizable set.

As an example, in Figure 19, state 1 is pre-stabilizable since disabling γ pre-stabilizes 1. However, disabling γ at state 2 leaves no other defined events at 2 and "kills" it, so neither state 2 or 3 is pre-stabilizable.

12.3.5 Stabilizability and (f,u)-Invariance

Stabilizability is an extension of pre-stabilizability. Stabilizability is described as finding a state feedback such that the closed loop system is stable. We can then define stabilizability formally as follows :

Given a live system A and some $E \subset X$, $x \in X$ is stabilizable with respect to E (or E-stabilizable) if there exists a state feedback K such that x is alive and E-stable in A_K. A set of states, Q, is a stabilizable set if there exists a feedback law $K(s)$ (a control pattern) so that every $x \in Q$ is alive and stable in A_K, and A is a stabilizable system if X is a stabilizable set.

In Figure 20, disabling β at state 2 is sufficient to make the whole system stable with respect to state 0. Disabling γ at state 1 will help stabilize only state 1, because the system can then continue looping between states 2 and 3. Disabling β at state 3 will not help stabilize or pre-stabilize any

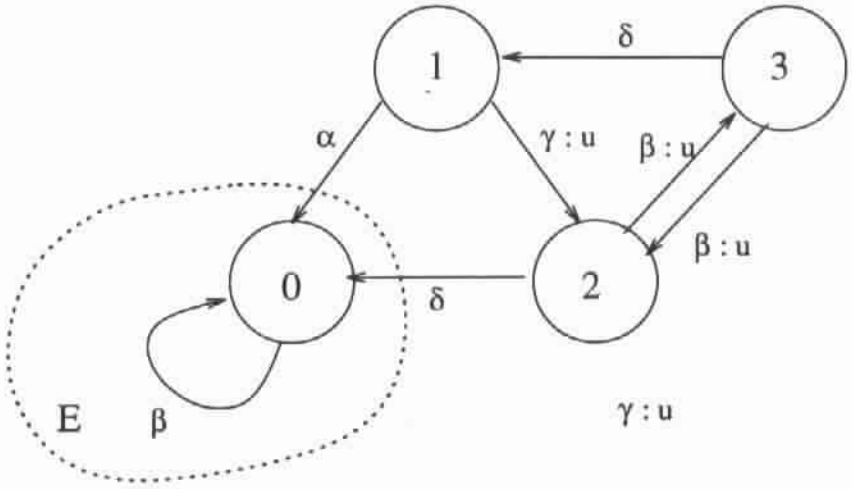


Figure 20: Stabilizability Example.

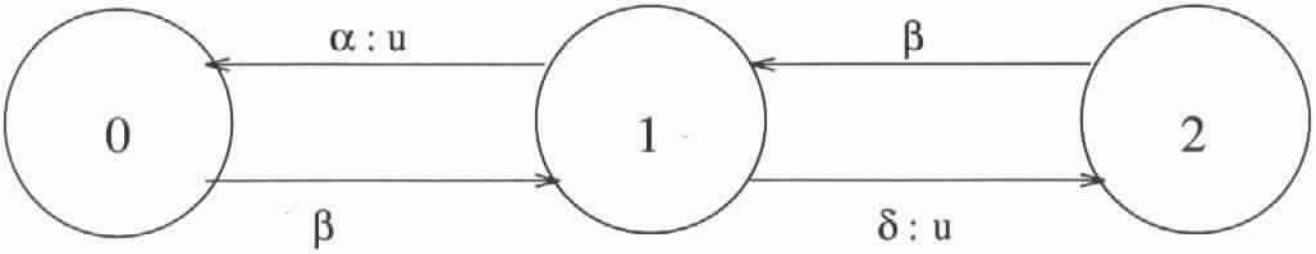


Figure 21: (f-u)-Invariance Example.

state.

Using control patterns to "drive" a subset of a system to be f-invariant is still another notion of stabilizability. A subset Q of X is (f,u)-invariant if there exists a state feedback K such that Q is f-invariant in A_K. Another notion of (f,u)-invariance is sustainable (f,u)-invariance. A subset Q of X is a sustainably (f,u)-invariant set if there exists a state feedback K such that Q is alive and f-invariant in A_K.

For example, in Figure 21, disabling event α at state 1 will make the subset $\{1, 2\}$ sustainably (f,u)-invariant. Also, disabling event δ at state 1 will make the subset $\{0, 1\}$ sustainably (f,u)-invariant.

12.4 Observability

In this section we address the problem of determining the current state of the system. In particular, we are interested in observing a certain sequence of *observable* events and making a decision regarding the state that the DEDS automaton A might possible be in. In our definition of observability, we visualize an intermittent observation model, no direct measurements of the state are made, the events we observe are only those that are in $\Gamma \subset \Sigma$, we will not observe events

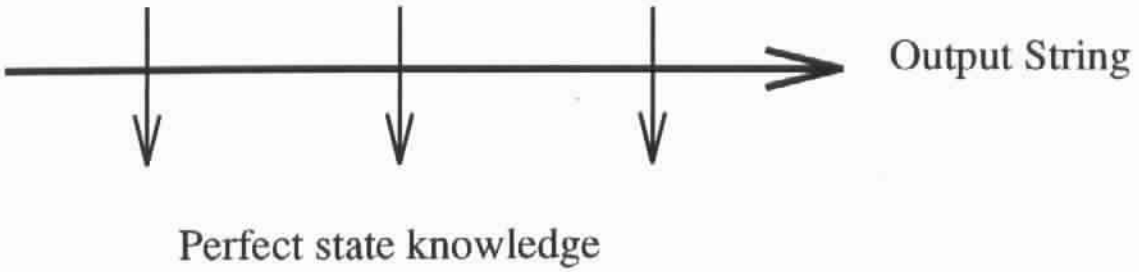


Figure 22: Notion of Observability: The state is known perfectly only at the indicated instants. Ambiguity may develop between these but is resolved in a bounded number of steps.

in $\Sigma \cap \overline{\Gamma}$ and will not even know that any of which has occurred. State ambiguities are allowed to develop (which must happen if $\Sigma \neq \Gamma$) but they are required to be resolvable after a *bounded* interval of events. This notion of observability can be illustrated graphically as in Figure 22.

12.4.1 Requirements

In developing the theory and examples we shall concentrate on uncontrolled models of DEDS automatons with partial knowledge of the event trajectory. Due to the fact that we are "seeing" only observable events in Γ in our system, it is not desirable to have our automaton generate arbitrarily long sequences of unobservable events in $\Sigma \cap \overline{\Gamma}$. A necessary condition to guarantee this is that the automaton after removing the observable events $A|\overline{\Gamma}$, must <u>not</u> be alive. In fact, it is also essential that every trajectory in $A|\overline{\Gamma}$ is killed in finite time by being forced into a dead state. It can be seen that the condition for a DEDS automaton to be unable to generate arbitrarily long sequences of unobservable events, is that $A|\overline{\Gamma}$ must be D-stable, where D is the set of states that only have observable events defined (i.e, $D = \{x \in X | d(x) \cap \overline{\Gamma}\}$).

12.4.2 State Observability

As illustrated in Figure 22, a DEDS is termed *observable* if we can use the observation sequence to determine the current state exactly at intermittent points in time separated by a *bounded* number of events. More formally, taking any sufficiently long string, s, that can be generated from any initial state x. For any observable system, we can then find a prefix p of s such that p takes x to a *unique* state y and the length of the remaining suffix is bounded by some integer n_o. Also, for any other string t, from some initial state x', such that t has the same output string as p, we require that t takes x' to the same, unique state y.

In Figures 23 and 24 a simple system and its observer are illustrated. It can be seen that the observer will never know when will the system be in states 3, 4 or 5, since the events that takes the system to those states are unobservable (δ/ϵ means that $\delta \in \Sigma \cap \overline{\Gamma}$), namely δ and γ. There are two states in the observer which are ambiguous, however, another two states are singleton states, i.e, when our observer reaches them, we'll know the exact state that the DEDS in currently in.

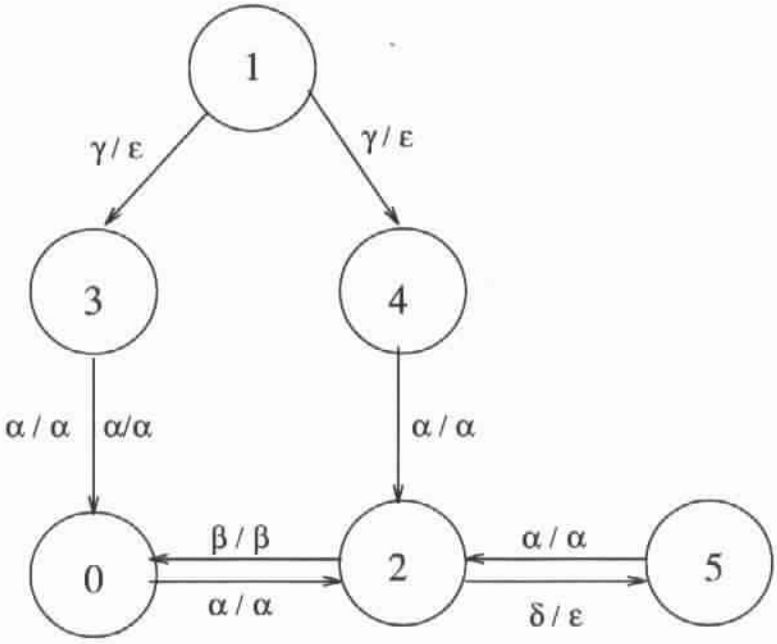


Figure 23: A Simple DEDS automaton.

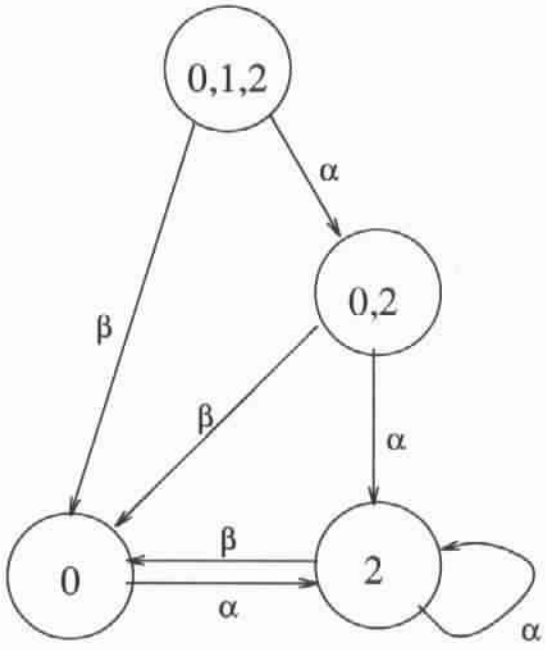


Figure 24: Observer for the simple DEDS automaton.

Had it been the case that our observer could, for example, loop forever in ambiguous states, then the DEDS would be unobservable. This leads to the following formal definition of observability that ties it with the notion of stability :

A DEDS automaton A is observable iff E is nonempty and O is E-stable.

where O is the observer for A and E is the set of singleton states of O. It can be seen that the observer in Figure 24 is stable with respect to the nonempty subset of states $\{0, 2\}$ and thus the DEDS of Figure 23 is observable.

12.4.3 Indistinguishability

We term pair of states (x, y) indistinguishable if they share an infinite length output (observable) event sequence. If we define :

$$Y_0 = \{x \in X |\ \nexists y \in X, \gamma \in \Sigma, \text{ such that } x \in f(y, \gamma)\}$$
$$Y_1 = \{x \in X | \exists y \in X, \gamma \in \Gamma, \text{ such that } x \in f(y, \gamma)\}$$
$$Y = Y_0 \cup Y_1$$

Then Y is the set of states x such that either there exists an observable transition defined from some state y to x, or x has no transition defined to it. As discussed above, the observer only uses the states in Y, and thus we can formally define indistinguishability for states in Y as follows :

Given $x \in X$, let $L_\infty(A, x)$ denote the set of infinite length event trajectories generated from x, and $h(L_\infty(A, x))$ the corresponding set of output (observable) trajectories. The pair $(x, y) \in Y \times Y$ is an indistinguishable pair if $h(L_\infty(A, x)) \cap h(L_\infty(A, y)) \neq \phi$.

It can be noticed that in Figure 23, (0,2) is an indistinguishable pair since an infinite string of α's is one of the possible observable output event sequences from either states. However, this system was shown to be observable, thus the non-existence of indistinguishabilities is not required for observability. If there are indistinguishable states, we will not always be able to determine which of these we were in at some point in the past, but this does not rule out the possibility that we may occasionally know the current state.

12.4.4 WD Observability

A system is termed WD observable if it is observable with a delay. It is required that there is perfect knowledge of the state some finite number of transitions into the past at intermittent points in time. Figure 25 illustrates the concept of WD observability.

As an example of a WD observable DEDS, Figure 26 represent such an automaton and its observer. All events in this example are assumed to be observable. The system is not observable since the observer does not have any singleton states (E is empty). When α or β occurs, we do not have perfect knowledge of the current state, but when either α or β happens we know perfectly what was the *previous* state.

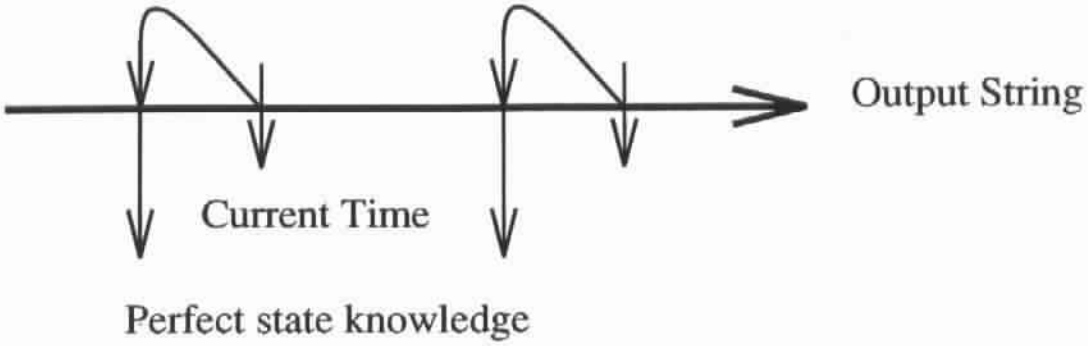


Figure 25: Observability with a Delay: The state, a finite number of transitions into the past, is known perfectly at intermittent (but not necessarily fixed) points in time.

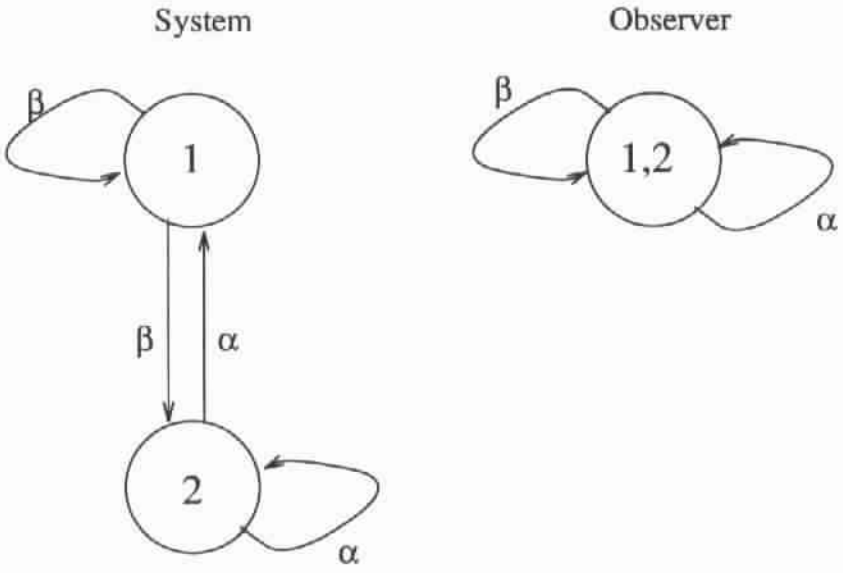


Figure 26: Example for WD Observability.

12.5 Output Feedback Stabilizability

In this section we combine the ideas discussed in the previous two sections regarding observability and stability to address the problem of stabilization by dynamic output feedback under partial observations. In this section we concentrate on partially controlled systems with partial knowledge of the event trajectory. In particular, the goal is to develop stabilizing compensators by cascading and a stabilizing state feedback defined on the observer's state space.

12.5.1 Requirements

To attack the problem of output feedback stabilization, it should be noticed that we are actually trying to "manipulate" the system's observer, in other words, what we have available in a sequence of observable events (the system's *output*) and we are trying to use this output to control the behavior of the system using *only* the events that we can control. It is then possible to redefine the problem of output feedback stabilization as the stabilization of the observer by state feedback.

The obvious notion of output E-stabilizability (stabilizability with respect to $E \subset X$) is the existence of a compensator C so that the closed-loop system A_C is E-stable. It is possible that such a stabilizing compensator exists, such that we are sure that the system passes through the subset E infinitely often (E-stable) *but* we never know when the system is in E. A stronger notion of output feedback stabilizability would not only requires that the system passes through subset E infinitely often, but also that we regularly know when the system is in E. In out example and discussion we shall concentrate on this stronger notion of output stabilizability.

12.5.2 Strong Output Stabilizability

The basic idea behind strong output stabilizability is that we will know that the system is in state E iff the observer state is a subset of E. The fact that the observer state should be a subset of E instead of having the observer state of interest *includes* states in E is because we want to *guarantee* that our system in within E. Our compensator should then *force* the observer to a state corresponding to a subset of E at intervals of at most a finite integer i observable transitions. We can then formalize the notion of a strongly output stabilizable system as follows :

A is strongly output E-stabilizable if there exists a state feedback K for the observer O such that O_K is stable with respect to $E_O = \{ \hat{x} \in Z \mid \hat{x} \subset E \}$.

where Z is the set of states of the observer.

As an example, considering the DEDS and its observer in Figure 27, where $E = \{1, 2\}$, we have to check the observer stability (or stabilize the observer) with respect to E_O, because this is the only observer state that is a subset of E. As a start, we do not know which state is our system in (as denoted by the state $\{0, 1, 2, 3\}$), however, using the observer transitions we can see that to achieve E_O-stability for the observer we only need to disable α at the observer state $\{0, 2\}$. It should be noted that all the events are observable in this DEDS automaton.

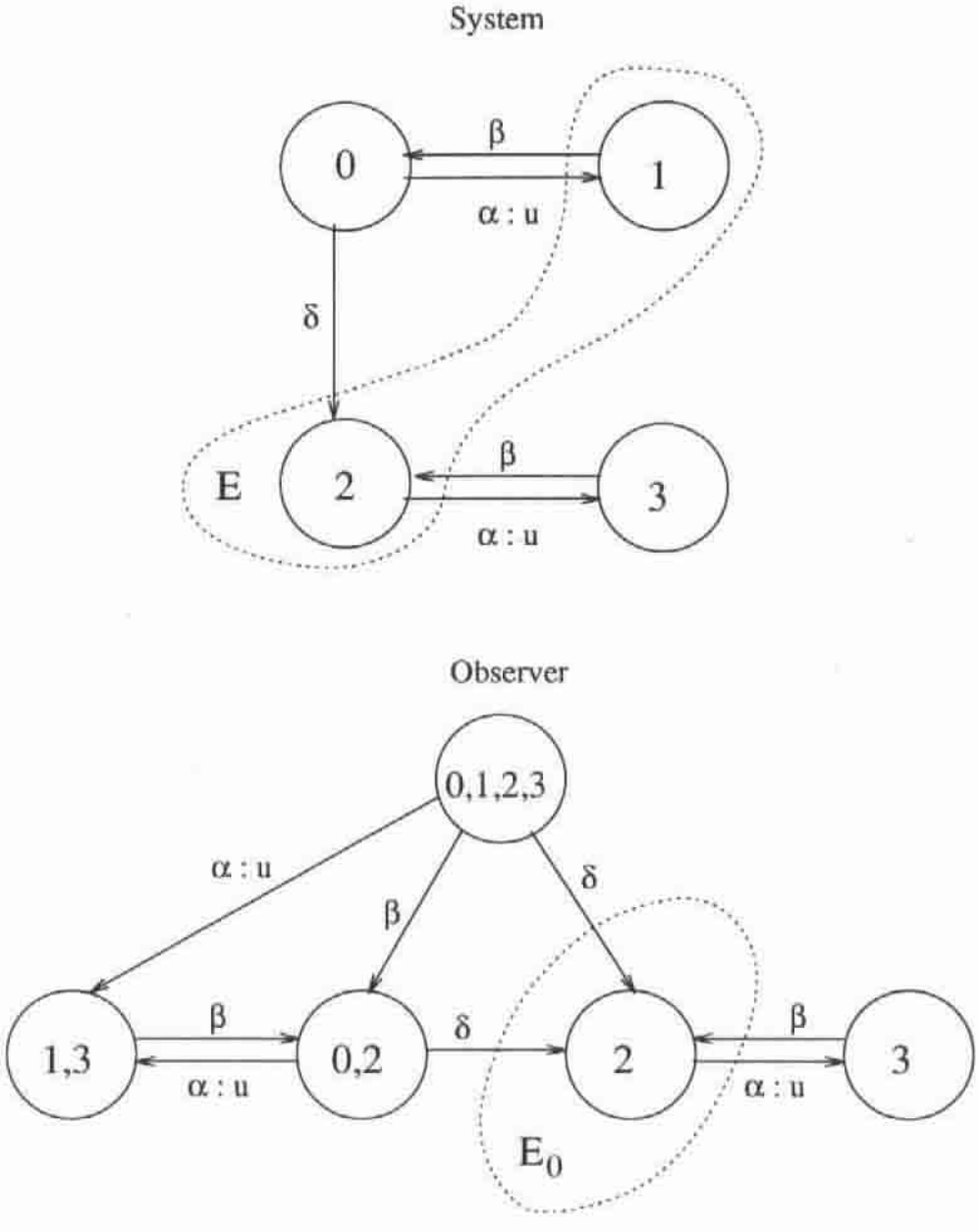


Figure 27: Example for Strong Output Stability (all the events are observable).

12.6 Invertibility

In this section we will discuss the notion of invertibility. The problem of invertibility arises due to the fact that a DEDS is, in general, a *partially* observable system. That is, "seeing" some events while observing a system does not imply that those events were the only ones that actually happened. The problem of reconstructing the *full* event sequence given only output (observable) events is what we term the invertibility problem.

12.6.1 Requirements

In order to be able to tackle this problem we need to use the automata model of a DEDS, so it will be assumed that the model of the DEDS behavior is known a-priori. The invertibility problem arises due to the fact that $\Gamma \subset \Sigma$, had $\Gamma = \Sigma$, invertibility would have been a trivial problem. The model we shall use in this section will be the standard model of a DEDS discussed in section 2, with partial knowledge of the event trajectory, however, the assumption that the system is uncontrollable will be made to simplify developing the theory. There are two notions of invertibility : The first notion assumes that the initial state in the DEDS automaton in known, the second notion does not assume that. It should be quite clear that the second notion will be harder to analyze, because it involves estimating the current state first. In our treatment of the problem we will discuss the first notion.

12.6.2 WD-Invertibility

By WD-invertibility we mean invertibility with a delay. We consider the DEDS automaton A that is the minimal automaton generating the event language $L = L(A, x_0)$, so that all the states can be reached from x_0, and no two states generate the same language. We also assume that A is deterministic. It should be "safe" enough to make those assumptions, because we will be concerned with the estimation of elements in L, we also can always choose a minimal deterministic automaton with an initial state that generates L, due to that fact that L will always be a regular language.

In particular, we are concerned with the problem that given L (or A and x_0), whether we can reconstruct an event trajectory $s \in L$ when we only observe the part of s in Γ.

We define a WD-invertible language, as one in which we can, at any time, use knowledge of the output (observable) sequence up to that time to reconstruct the full event sequence up to a point *at most* an integer number of events n_d into the past.

Figure 28 shows a graphical explanation of the notion of WD-invertibility.

WD-invertibility can be illustrated by an example as in Figure 29. In this system, state 0 in the initial state. The notation α/ϵ means that event α is not observable. In this case, L is WD-invertible with $n_d = 4$. It is not invertible *at all* without delay (i.e, with $n_d = 0$). As an example, if we observe σ^2, the original input sequence could be $\alpha(\delta\sigma)^2$ or $\alpha(\delta\sigma)^2\delta$ or $\alpha\delta\sigma\sigma$, etc., but the first three events are known with certainty.

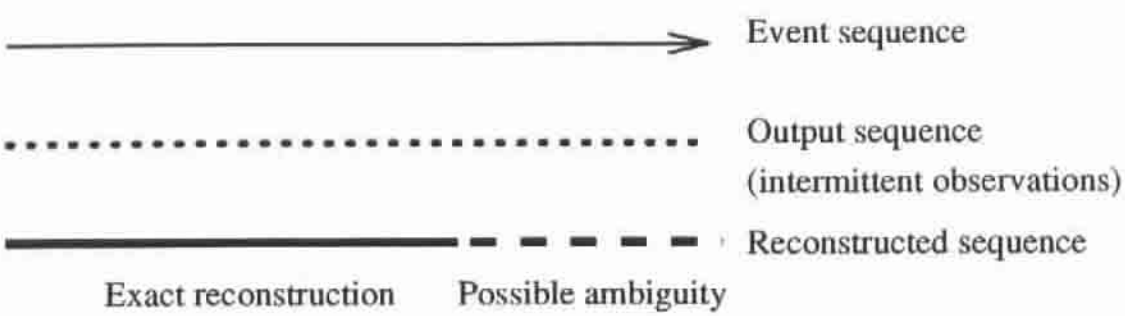


Figure 28: Invertibility with a Delay: Given the output sequence, the event sequence is reconstructed exactly but with some delay. The ambiguity at the end of the reconstructed string will be resolved using future observations.

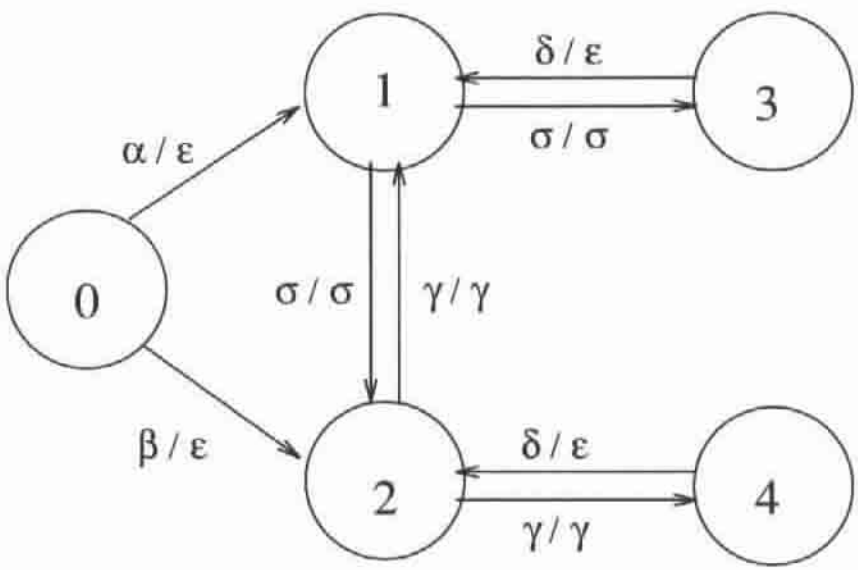


Figure 29: Example for WD-Invertibility: State 0 is the initial state.

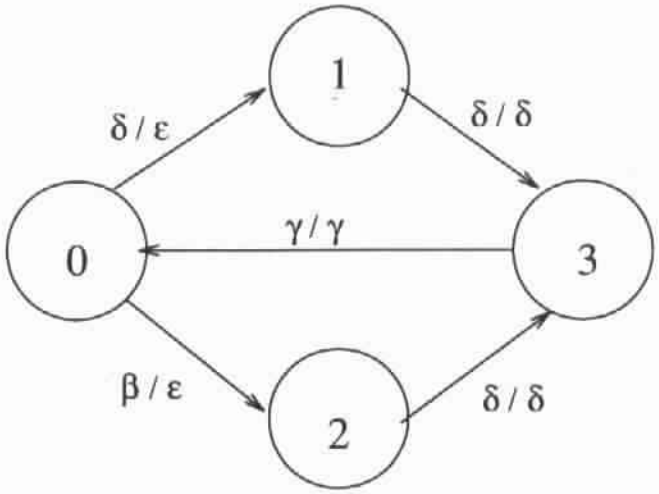


Figure 30: Example for an Ambiguous System.

12.6.3 Ambiguity and Non-Invertible DEDS

To discuss the notions of ambiguity and non-invertibility we need to define a few notations on languages. In particular :

- $L_f(A,x)$: All the strings in $L_(A,x)$ with observable events as their last events.
- $L_1(A,x)$: Those strings in $L_f(A,x)$ that have only one observable event.
- $L_\sigma(A,x)$: The set of strings in $L_1(A,x)$ that have $\sigma \in \Gamma$ as the observable event.

A DEDS automaton A is termed ambiguous if for some $x \in X$ and $\gamma \in \Gamma$, there exists distinct strings $s,t \in L_\gamma(A,x)$ such that $f(x,s) = f(x,t)$. Moreover, if A ambiguous, then L is not WD-invertible. In other words, if there exists two different sequences of events taking a state to another, and with the same observable event for both sequences, then the language is not WD-invertible. This is because no future behavior will enable us to distinguish between those strings.

In Figure 30, the system is ambiguous as both $\alpha\delta$ and $\beta\delta$, which produce the same output (observable events), take state 0 to state 3. Thus the language generated from state 0 is not invertible.

A DEDS automaton A might be non-invertible although it is unambiguous, that is, unambiguity alone is not sufficient for invertibility. For example, the automaton in Figure Figure 31, where 0 is the initial state, is not ambiguous, but L is not invertible, since the event trajectories $(\beta\alpha)^*$ and $(\delta\alpha)^*$ both have the same output α^*. Following from the fact that $R(A,x_0) = X$, one can say that L is WD-invertible iff $L(A,x)$ is WD-invertible for each $x \in X$.

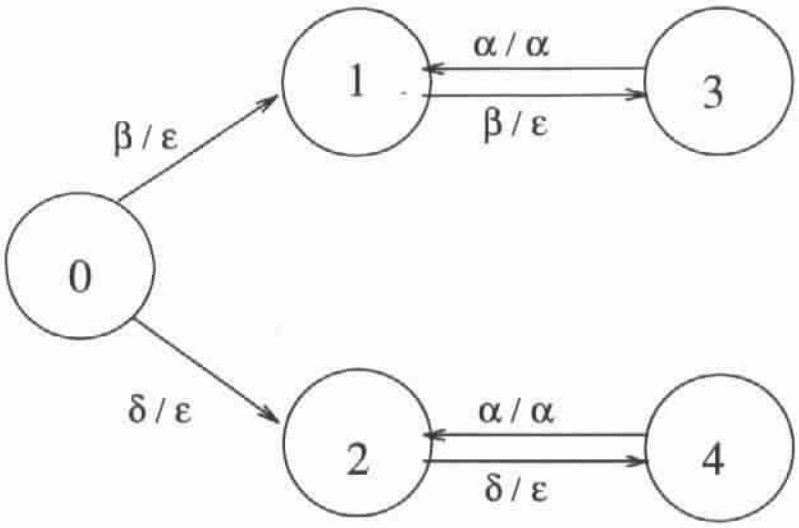


Figure 31: Example for an Unambiguous but not Invertible System: State 0 is the initial state.

12.7 Discussion and Future Work

In this section, we have reviewed some basic notions related to discrete event dynamic systems. We emphasized upon the automaton model of a DEDS and described some ideas regarding controlling and observing the behavior of such systems. As a future extension, more powerful models could be used instead of finite automata, for example, Grammars, Pushdown Automata, Turing Machines and/or μ-recursive functions. Applications related to fields other than communication and manufacturing systems could be exploited. Many dynamic tasks can be modeled as DEDS and thus they can be analyzed and controlled efficiently using the ideas discussed in this section.

13 Proposed Model

We have built a software environment to aid in the design, analysis and simulation of Discrete Event and Hybrid Systems. The environment allows the user to build a system using either Finite State Machines or Petri-Nets. The environment runs under X/Motif and supports a graphical DES (Discrete Event System) hybrid controller, simulator, and analysis framework. The framework allows for the control, simulation and monitoring of dynamic systems that exhibits a combination of symbolic, continuous, discrete, and chaotic behaviors, and includes stochastic timing descriptions (for events, states, and computation time), probabilistic transitions, controllability and observability definitions, temporal, timed, state space, petri-nets, and recursive representations, analysis, and synthesis algorithms. The environment allows not only the graphical construction and mathematical analysis of various timing paths and control structures, but also produces C code to be used as a controller for the system under consideration.

Using the environment is fairly simple. For finite state machines the designer uses the mouse to place states (represented by ovals) and connect them with events (represented by arrows).

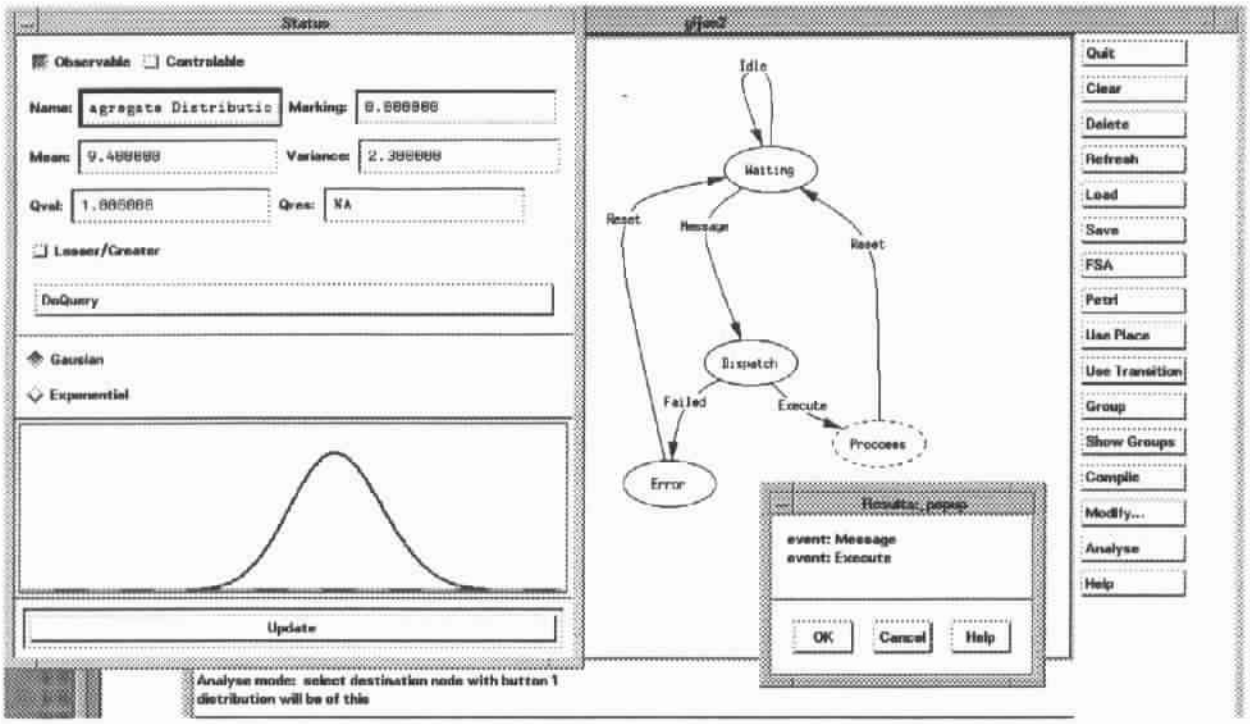


Figure 32: A stochastically timed FSM window during analysis

Transitions and states can be added, moved and deleted easily. Figure 32 is an example of a simple stochastically timed FSM, containing 4 states and 5 events.

The probabilities on the events (that is, which path to navigate in the automaton) is designated using the mark field in the status dialog box. The different timings (on event and state times) and distribution function type, mean and variance can be assigned through the status dialog box too. The allowable distributions are currently restricted to Gaussian and exponential functions, but can be easily extended to arbitrary discrete or continuous distributions. A window shows the distribution function at a state or event, and also allows the user to do queries. For example: queries on whether a path time probability is greater or less than a give time, or combined timing distributions to reach a goal state through various paths, etc. The dialog box allows the user to perform queries of various kinds. The currently selected state/event is drawn with a dashed line, and the information in the status window pertains to it. Optimizing paths based on stochastic timing can also be performed, in that case, windows will pop out with the event path, and the status window will have the combined distribution function. Figure 33 presents an automaton model in the environment. The environment also produces C code for controlling the system under consideration.

In our PN model we have extended the definition of stochastic timed Petri Nets, to have additional timings. Our model has three times associated with it, a place time, a delay time, and an event time (see Figure 34). The place time is a time where the token is held back, and delays the enabling of the transition, this represents the computation time of that place. The

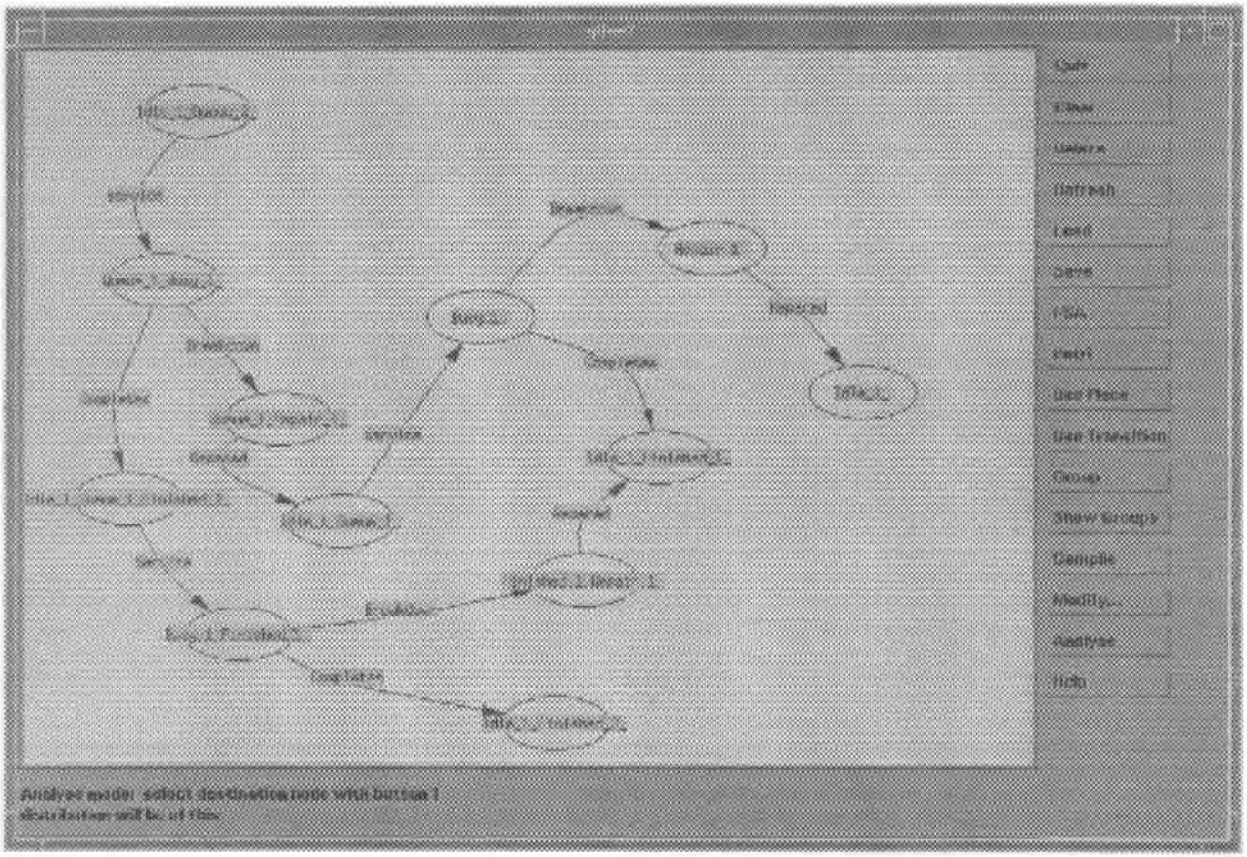


Figure 33: A snap shot of the FSM environment

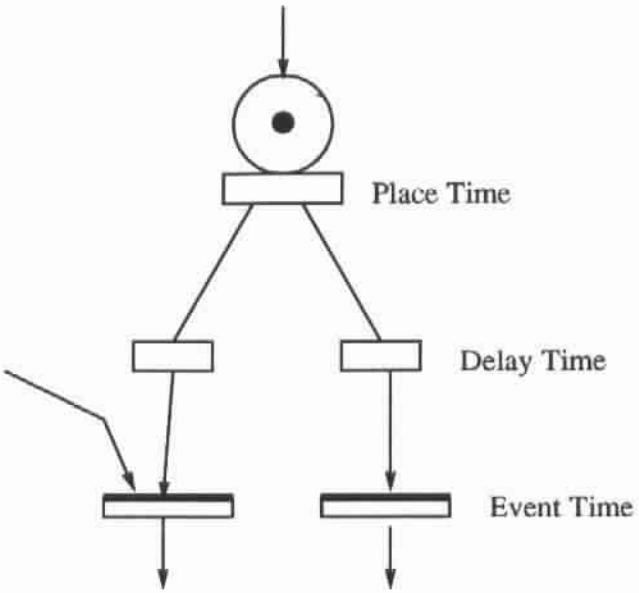


Figure 34: The proposed three time zones for a timed Petri net.

delay time is a time associated with the input arcs to a transition, it represents the time to leave the corresponding place. The event time is analogous to the single time in stochastic timed Petri Nets which is called *firing time*. We believe that this lends to a more intuitive representation of the times, and simplifies the modeling task since it captures more details than the original timed Petri net model.

We can define the new model as:

$$PN = (P, T, A, W, x_0)$$

where,

- P = set of places with associated random variables
- T = set of transitions
- $A = A_{in} \cup A_{out}$ with
  - A_{in} set of elements from $\{P \times T\}$ with associated random variables
  - A_{out} set of elements from $\{T \times P\}$
- W = a weight function, $w : A \rightarrow \{1, 2, 3, \ldots\}$
- x_0 is an initial marking

The environment for Petri-Nets is similar. Places are represented graphically by circles, transitions by ellipses, and arcs by arrows. As mentioned above, there are three locations where one can place timing information, on the events - the event time, which is the time the actual event

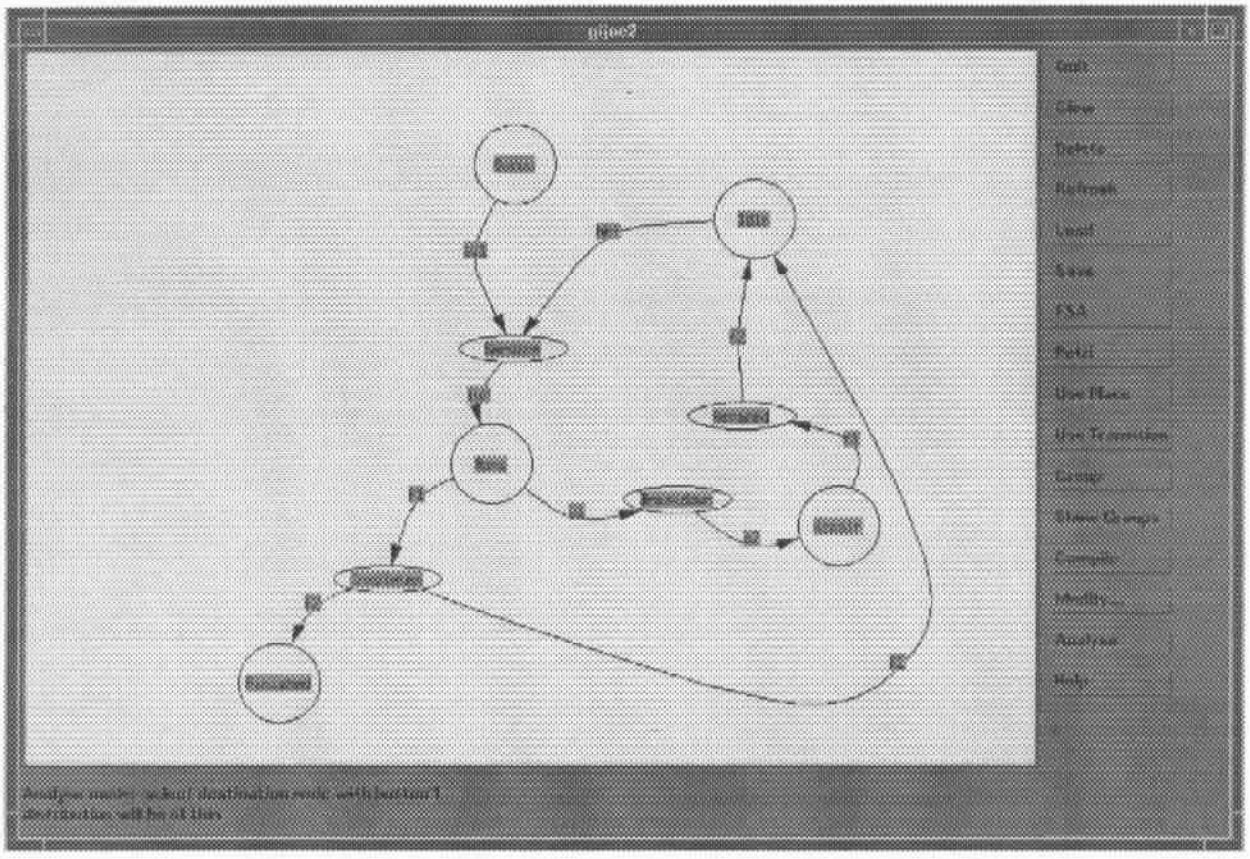


Figure 35: A snap shot of the Petri-net environment

takes, place time - when a token is moved, through a transition firing, there is a place time, which hides the token until it has expired, the final time is a delay time, this comes into effect when a transition fires, it is the time for the event to reach the transition, the event time will not start until all of its input tokens delay time has expired. Figure 35 depicts a snap shot of the Petri-Net environment in action.

The system generates C code for the user hybrid system, so one can simulate and control an actual system using the code. The C code is currently generated for FSMs (soon code will be generated for PN's too). A Petri Net will be converted to a FSM before code is generated, all of the timing is then placed on the events. The user has to select the initial state, and provide the function for simulating/generating the events, the code will keep track of the elapsed simulated time, and will return when it reaches a state with no transitions.

The environment allows conversion back and forth between the FSM and PN models. Conversion to a Petri net is straight forward, but one looses the event probabilities. The only thing that's needed is to create a transition for every event. Conversion from a Petri-net to a FSM is only possible if the PN is k-bounded, which means no place can ever have more than k tokens. The system generates a state for every possible marking of that net. The states are represented as the marking, the events are just the transitions. Three 3 times are pushed into the events,

The system convolves the maximum of the input delays, with the event, and the maximum of the place times. The maximum function is a standard convolution, except that the maximum is used instead of multiplication.

The algorithm for generating all of the markings starts with some initial marking, then goes through all of the possible transitions, if it can fire, the firing is simulated, and the new marking is inserted in the set of states, if it is already represented, the transition is kept; otherwise the transition is kept and recursion is done with the new marking. This process is repeated till no transitions can be fired.

Our system serves as much-needed graphical simulator, analyzer, synthesizer, monitor, and controller for complex hybrid systems models using either Petri nets or FSMs high-level frameworks.

14 Discrete Event Observation Under Uncertainty

We present a new framework and representation for the general problem of observation. The system being studied can be considered as a "hybrid" one, due to the fact that we need to report on *distinct* and *discrete* visual states that occur in the *continuous*, *asynchronous* and three-dimensional world, from two-dimensional observations that are sampled periodically. In other word, the system being observed and reported on consists of a number of continuous, discrete and symbolic parameters that vary over time in a manner that might not be "smooth" enough for the observer, due to visual obscurities and other perceptual uncertainties.

The problem of observing a moving agent was addressed in the literature extensively. It was discussed in the work addressing tracking of targets and, determination of the optic flow [7,36,102,197], recovering 3-D parameters of different kinds of surfaces [21,134,190,194], and also in the context of other problems [6,13,48,84]. However, the need to *recognize*, *understand* and *report* on different visual steps within a dynamic task was not sufficiently addressed. In particular, there is a need for high-level symbolic interpretations of the actions of an agent that attaches meaning to the 3-D world events, as opposed to simple recovery of 3-D parameters and the consequent tracking movements to compensate their variation over time.

In this work we establish a framework for the general problem of observation, recognition and understanding of dynamic visual systems, which may be applied to different kinds of visual tasks. We concentrate on the problem of observing a manipulation process in order to illustrate the ideas and motive behind our framework. We use a discrete event dynamic system as a high-level structuring technique to model the visual manipulation system. Our formulation uses the knowledge about the system and the different actions in order to solve the observer problem in an efficient, stable and practical way. The model incorporates different hand/object relationships and the possible errors in the manipulation actions. It also uses different tracking mechanisms so that the observer can keep track of the workspace of the manipulating robot. A framework is developed for the hand/object interaction over time and a stabilizing observer is constructed.

Low-level modules are developed for recognizing the "events" that causes state transitions within the dynamic manipulation system. The process uses a coarse quantization of the manipulation actions in order to attain an active, adaptive and goal-directed sensing mechanism.

The work examines closely the possibilities for errors, mistakes and uncertainties in the visual manipulation system, observer construction process and event identification mechanisms, leading to a DEDS formulation with uncertainties, in which state transitions and event identification is asserted according to a computed set of 3-D uncertainty models.

We motivate and describe a DEDS automaton model for visual observation in the next section and then proceed to formulate our framework for the manipulation process and the observer construction. Then we develop efficient low-level event-identification mechanisms for determining different manipulation movements in the system and for moving the observer. Next, the uncertainty levels are discussed. Some results from testing the system are enclosed.

14.1 Hybrid and Discrete Event Dynamic Systems for Robotic Observation

Hybrid systems, in which digital and analogue devices and sensors interact over time, is attracting the attention of researchers. Representation of states and the physical system condition includes continuous and discrete numerics, in addition to symbols and logical parameters. Most of the current vision and robotics problems, as well as problems in other domains, fall within the description of hybrid systems. There as many issues that need to be resolved, among them, definitions for observability, stability and stabilizability, controllability in general, uncertainty of state transitions and identification of the system. The general observation problem falls within the hybrid system domain, as there is a need to report, observe and control *distinct* and *discrete* system states. There is also a need for recognizing *continuous* 2-D and 3-D evolution of parameters. Also, there should be a *symbolic* description of the current state of the system, especially in the manipulation domain.

We do not intend to give a solution for the problem of defining, monitoring or controlling such hybrid systems in general. What we intend to present in this work is a framework that works for the class of hybrid systems encountered within the robotic observation paradigm. The representation we advocate allows for the symbolic and numeric, continuous and discrete aspects of the observation task. We conjecture that the framework could be explored further as a possible basis for providing solutions for general hybrid systems representation and analysis problems.

We suggest the use of a representation of discrete event dynamic systems, which is augmented by the use of a concrete definition for the events that causes state transitions, within the observation domain. We also use some uncertainty modeling to achieve robustness and smoothness in asserting state and continuous event variations over time.

Dynamic systems are sometimes modeled by finite state automata with partially observable events together with a mechanism for enabling and disabling a subset of state transitions [129,145,159], the reader is referred to those references for more information about this class of

DEDS representation. We propose that such a DEDS skeleton is a suitable high-level framework for many vision and robotics tasks, in particular, we use the DEDS model as a high-level structuring technique for a system to observe a robot hand manipulating an object.

14.1.1 Discrete event dynamic systems for active visual sensing

An example of a high-level DEDS controller for part inspection can be seen in Figure 36. This finite state machine has some observable events that can be used to control the sequencing of the process. The machine remains in state A until a part is loaded. When the part is loaded, the machine transitions to state B where it remains until the part is inspected. If another part is available for inspection, the machine transitions to state A to load it. Otherwise, state C, the ending state, is reached. If an interruption occurs, such as a misloaded part or inspection error, the machine goes to state D, the error state.

Our approach uses DEDS to drive a semi-autonomous visual sensing module that is capable of making decisions about the *visual state* of the manipulation process taking place. This module provides both symbolic and parametric descriptions which can be used to observe the process *intelligently* and *actively*.

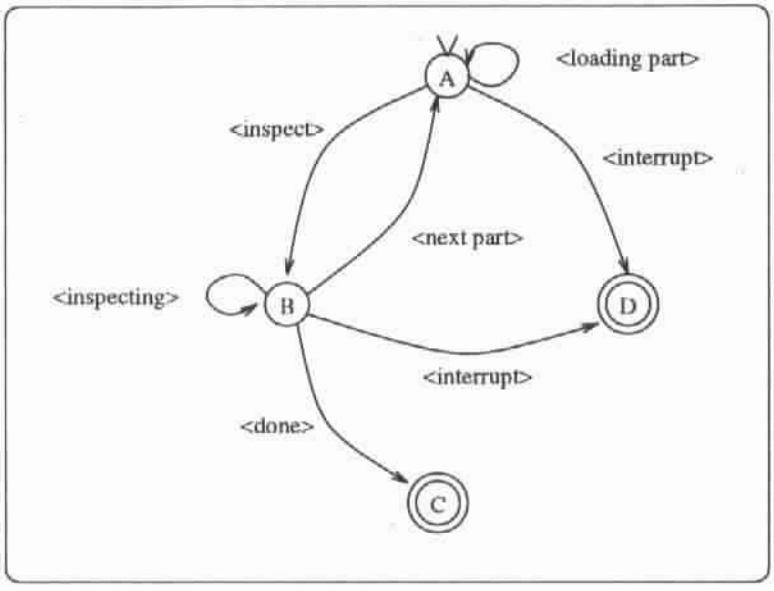


Figure 36: A Simple FSM

A DEDS framework is used to model the tasks that the autonomous observer system executes. This model is used as a high level structuring technique to preserve and make use of the information we know about the way in which a manipulation process should be performed. The state and event description is associated with different visual cues; for example, appearance of objects, specific 3-D movements and structures, interaction between the robot and objects, and occlusions. A DEDS observer serves as an intelligent sensing module that utilizes existing information

about the tasks and the environment to make informed tracking and correction movements and autonomous decisions regarding the state of the system.

To be able to determine the current state of the system we need to observe the sequence of events occurring in the system and make decisions regarding the state of the automaton. State ambiguities are allowed to occur, however, they are required to be resolvable after a bounded interval of events. In a *strongly output stabilizable* system, the state of the system is known at bounded intervals and allowable events can be controlled (enabled or disabled) in a way that ensures return in a bounded interval to one of a desired and known set of states (visual states in our case).

One of the objectives is to make the system strongly output stabilizable and/or construct an observer to satisfy specific task-oriented visual requirements. Many 2-D visual cues for estimating 3-D world behavior can be used. Examples include: image motion, shadows, color and boundary information. The uncertainty in the sensor acquisition procedure and in the image processing mechanisms should be taken into consideration to compute the world uncertainty.

The observer framework can be utilized for recognizing error states and sequences. This recognition task will be used to report on *visually incorrect* sequences. In particular, if there is a pre-determined observer model of a particular manipulation task under observation, then it would be useful to determine if something goes wrong with the exploration actions. The goal of this reporting procedure is to alert the operator or autonomously supply feedback to the manipulating robot so that it can correct its actions.

14.1.2 DEDS for Modeling Observers

DEDS can be considered as very suitable tools for modeling observers. In particular, in the manipulation observer domain, there is a need to recognize and report on distinct and discrete visual states, which might represent manipulation tasks and/or sub-tasks. The observer should have the ability to state a symbolic description of the current manipulation agent action. The coarse definition of DEDS states provide a means for such symbolic state descriptions.

The definition for observers and the observer construction process for discrete event systems are very coherent with the requirements for an autonomous robotic observer. The purpose of DEDS observers is to be able to reconstruct the system state, which is exactly the requirements for a visual observer, which needs to recognize, report and possibly act, depending on the visual manipulation state. The notions of controllable actions is easily mapped to some tracking and repositioning procedures that the robotic observer will have to undertake in order to "see" the scene from the "best" viewing position as the agent under observation moves over time. The actions which the observer robot might need to perform, depends on the sequence of "observable" events and the reconstructed state path.

Event description in a visual observer is possibly a combination of different 2-D and 3-D visual data. The visual primitives used in an observer domain could be motion primitives, matching measures, object identification processes, structure and shape parameters and/or a number of other visual cues. The problem with the DEDS skeleton is that it does not allow for smooth state changes under uncertainty in recovering the events. We describe in the next sections techniques

that make the transition from a DEDS skeleton into a working hybrid observer for a moving manipulation agent. Stability and stabilizability issues are resolved in the visual observer domain by supplying suitable control sequences to the observer robot at intermittent points in time in order to "guide" it into the "desirable" set of visual states.

14.2 State Modeling and Observer Construction

Manipulation actions can be modeled efficiently within a discrete event dynamic system framework. It should be noted that we do not intend to *discretize* the workspace of the manipulating robot hand or the movement of the hand, we are merely using the DEDS model as a high level structuring technique to preserve and make use of the information we know about the way in which each manipulation task should be performed, in addition to the knowledge about the physical limitations of both the observer and manipulating robots. The high-level state definition permits the observer recognize and report on symbolic descriptions of the task and the physical relationships under observation. We avoid the excessive use of decision structures and exhaustive searches when observing the 3-D world motion and structure.

A bare-bone approach to solving the observation problem would have been to try and visually reconstruct the full 3-D motion parameters of the robot hand, which would have more than six degrees of freedom, depending on the number of fingers and/or claws and how they move. The motion and shape or structure of the different objects should also be recovered in 3-D, which is complicated especially if some of them are non-rigid bodies. That process should be done in real time while the task is being performed. A simple way of tracking might be to try and keep a fixed geometric relationship between the observer camera and the hand over time. However, the above formulation is inefficient, unnecessary and for all practical purposes infeasible to compute in real time. In addition, that formulation does not provide any kind of interpretation for the *meaning* of the scene evolution, nor does it allow for any symbolic recognition for the task under observation. The limitation of the observer reachability and the extensive computations required to perform the visual processing are motives behind formulating the problem as a hierarchy of task-oriented observation modules that exploits the higher-level knowledge about the existing system, in order to achieve a feasible mechanism of keeping the visual process under supervision.

14.2.1 State Space Modeling

We do a coarse quantization of the *visual manipulation actions* which allows modeling both continuous and discrete aspects of the manipulation dynamics. State transitions within the manipulation domain are asserted according to probabilistic models that determine at different instances of time whether the visual scene under inspection has changed its state within the discrete event dynamic system state space. Mapping the desired visual states to a DEDS skeleton is a straight forward procedure. We attach a DEDS automaton state to each meaningful visual state within a manipulation action. The quantization threshold depends on the application requirement. In other words, the state space can be expanded or contracted depending on the level of accuracy required in reporting and observing. A surgical operation step, performed by a robotic end ef-

fector, will obviously require an observer that reports (and possibly control the effector within a closed-loop visual system) with extreme precision. The observer for a robotic manipulator whose task is to pile up heaps of waste would, most likely, report in a crude fashion, thus needing a small number of states. The quantization threshold depends heavily on the nature of the task and the application requirements. The DEDS formulation is flexible, in the sense that it allows different precisions and/or state space models depending on the requirements.

The task of building DEDS automaton skeletons for observer agents can be performed either *manually* or *automatically*. In the manual formation case, the designer would have to draw the automaton model that best suits the task(s) under observation and depending on the application requirements and implement the code for the state machine. Automatic construction of the state machine could be done by having a *learning* stage [117,118] in which a mapping module would form the automaton. This is performed before the actual observation process is invoked. The idea is to supply the module with sets of possible sequences in the form of *strings* of a certain language that the DEDS automaton should minimally accept. The language could be either supplied by an operator, in which case, the resulting automaton performance depends on the relative skill of the operator, or through showing the module a sequence of visual actions and labeling those actions appropriately. The language strings should also be accompanied by a set of transitional conditions as event descriptions. The module would then produce the minimal DEDS automaton, complete with event and state descriptions that accepts the language.

We next discuss building the manipulation model for some simple tasks, then we proceed to develop the observer for these tasks. Formulating the models for the state transitions, the inter-state continuous dynamics and recovering uncertainty will be left for sections 4 and 5 which deal with the different uncertainty levels and event identification mechanisms.

14.2.2 Building the Model

The ultimate goal of the observation mechanism is to be able to know at all (or most) of the time what is the current manipulation process and what is the visual relationship between the hand and the object. The fact that the observer will have to move in order to keep track of the manipulation process, makes one think of the stabilizability principle for general DEDS as a model for the tracking technique that has to be performed by the observer's camera.

In real-world applications, many manipulation tasks are performed by robots, including, but not limited to, lifting, pushing, pulling, grasping, squeezing, screwing and unscrewing of machine parts. Modeling all the possible tasks and also the possible order in which they are to be performed is possible to do within a DEDS state model. The different hand/object visual relationships for different tasks can be modeled as the set of states X. Movements of the hand and object, either as 2-D or 3-D motion vectors, and the positions of the hand within the image frame of the observer's camera can be thought of as the events set Γ that causes state transitions within the manipulation process. Assuming, for the time being, that we have no direct control over the manipulation process itself, we can define the set of admissible control inputs U as the possible tracking actions that can be performed by the hand holding the camera, which actually can alter the visual configuration of the manipulation process (with respect to the observer's camera).

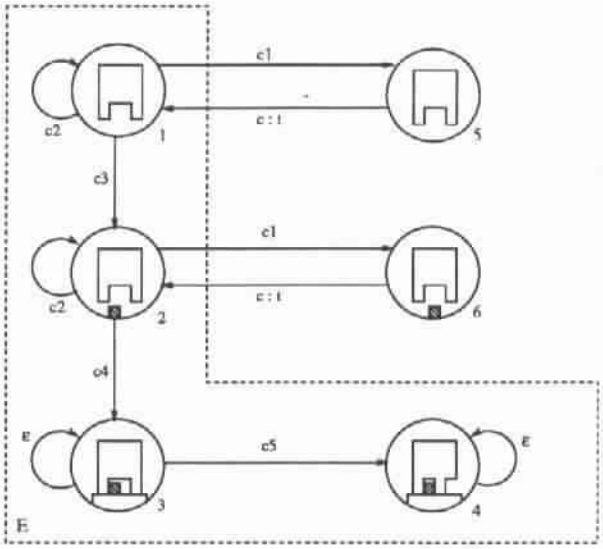


Figure 37: A Model for A Grasping Task

Further, we can define a set of "good" states, where the visual configuration of the manipulation process enables the camera to keep track and to know the movements in the system. Thus, it can be seen that the problem of observing the robot reduces to the problem of forming an output stabilizing observer (an observer that can always return to a set of "good" visual states) for the system under consideration.

It should be noted that a DEDS representation for a manipulation task is by no means unique, in fact, the degree of efficiency depends on the designer who builds the model for the task, testing the optimality of a visual manipulation models is an issue that remains to be addressed. Automating the process of building a model was discussed in the previous section. As the observer identifies the current state of a manipulation task in a non ambiguous manner, it can then start using a practical and efficient way to determine the next state within a predefined set, and consequently perform necessary tracking actions to stabilize the observation process with respect to the set of good states. That is, the current state of the system tells the observer what to *look for* in the next step.

- A Grasping Task

We present a simple model for a grasping task. The model is that of a gripper approaching an object and grasping it. The task domain was chosen for simplifying the idea of building a model for a manipulation task. It is obvious that more complicated models for grasping or other tasks can be built. The example shown here is for illustration purposes.

As shown in Figure 37, the model represents a view of the hand at state 1, with no object in sight, at state 2, the object starts to appear, at state 3, the object is in the claws of the gripper and at state 4, the claws of the gripper close on the object. The view as presented in the figure

is a frontal view with respect to the camera image plane, however, the hand can assume any 3-D orientation as so long as the claws of the gripper are within sight of the observer, for example, in the case of grasping an object resting on a tilted planar surface. This demonstrates the continuous dynamics aspects of the system. In other words, different orientations for the approaching hand are allowable and observable. State changes occur only when the object appear in sight or when the hand encloses it. The frontal upright view is used to facilitate drawing the automaton only. It should be noted that these states can be considered as the set of good states E, since these states are the expected different visual configurations of a hand and object within a grasping task.

States 5 and 6 represent instability in the system as they describe the situation where the hand is not centered with respect to the camera imaging plane, in other words, the hand and/or object are not in a good visual position with respect to the observer as they tend to escape the camera view. These states are considered as "bad" states as the system will go into a non-visual state unless we correct the viewing position. The set $X = \{1, 2, 3, 4, 5, 6\}$ is the finite set of states, the set $E = \{1, 2, 3, 4\}$ is the set of "good" states. Some of the events are defined as motion vectors or motion vector probability distributions, as will be described later, that causes state transitions and as the appearance of the object into the viewed scene. The transition from state 1 to state 2 is caused by the appearance of the object. The transition from state 2 to state 3 is caused by the event that the hand has enclosed the object, while the transition from state 3 to state 4 is caused by the inward movement of the gripper claws. The transition from the set $\{1, 2\}$ to the set $\{5, 6\}$ is caused by movement of the hand as it escapes the camera view or by the increase in depth between the camera and the viewed scene, that is, the hand moving far away from the camera. The self loops are caused by either the stationarity of the scene with respect to the viewer or by the continuous movement of the hand as it changes orientation but without tending to escape a good viewing position of the observer. In the next section we discus different techniques to identify the events. The controllable events denoted by "$: t$" are the tracking actions required by the hand holding the camera to compensate for the observed motion. Tracking techniques will later be addressed in detail. All the events in this automaton are observable and thus the system can be represented by the triple $G = (X, \Sigma, T)$, where X is the finite set of states, Σ is the finite set of possible events and T is the set of admissible tracking actions or controllable events.

It should be mentioned that this model of a grasping task could be extended to allow for error detection and recovery. Also search states could be added in order to "look" for the hand if it is no where in sight. The purpose of constructing the system is to develop an observer for the automaton which will enable the determination of the current state of the system at intermittent points in time and further more, enable us to use the sequence of events and control to "guide" the observer into the set of good states E and thus stabilize the observation process. Disabling the tracking events will obviously make the system unstable with respect to the set $E = \{1, 2, 3, 4\}$ (can't get back to it), however, it should be noted that the subset $\{3, 4\}$ is already stable with respect to E regardless of the tracking actions, that is, once the system is in state 3 or 4, it will remain in E. The whole system is stabilizable with respect to E, enabling the tracking events will cause all the paths from any state to go through E in a finite number of transitions and then will visit E infinitely often.

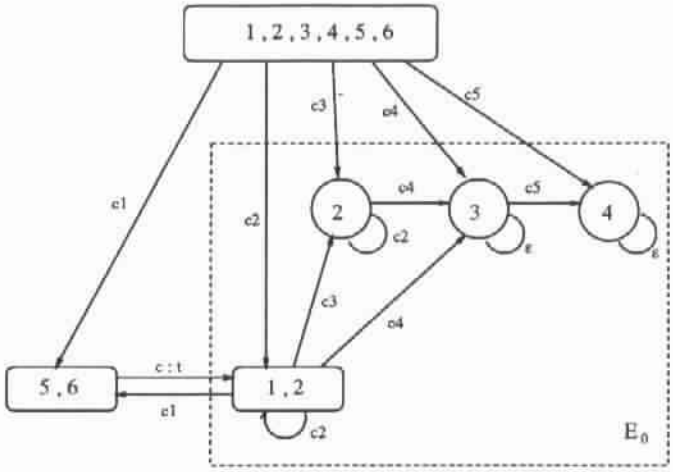


Figure 38: An Observer for the Grasping System

14.2.3 Developing the Observer

In order to know the current state of the manipulation process we need to observe the sequence of events occurring in the system and make decisions regarding the state of the automaton, state ambiguities are allowed to occur, however, they are required to be resolvable after a *bounded* interval of events. An observer, have to be constructed according to the visual system for which we developed a DEDS model. The goal will be to make the system a stabilizable one and/or construct an observer to satisfy specific task-oriented visual requirements that the user may specify depending on the nature of the process. It should be noticed that events can be asserted with a specific probability as will be described in the sections to come and thus state transitions can be made according to pre-specified thresholds that compliments each state definition. In the case of developing ambiguities in determining current and future states, the history of evolution of past event probabilities can be used to navigate backwards in the observer automaton till a strong match is perceived, a fail state is reached or the initial ambiguity is asserted.

As an example, for the model of the grasping task, an observer can be formed for the system as shown in Figure 38. It can be easily seen that the system can be made stable with respect to the set E_O (The system always returns to that set).

At the beginning, the state of the system is totally ambiguous, however, the observer can be "guided" to the set E_O consisting of all the subsets of the good states E as defined on the visual system model. It can be seen that by enabling the tracking event from the state (5, 6) to the state (1, 2), all the system can be made stable with respect to E_O. The singleton states represent the instances in time where the observer will be able to determine without ambiguity the current state of the system.

In the next section we shall elaborate on defining the different events in the visual manipulation system and discuss different techniques for event and state identification. We shall also introduce a framework for computing the uncertainty in determining the observable visual events in the system and a method by which the uncertainty distribution in the system can be used to efficiently keep

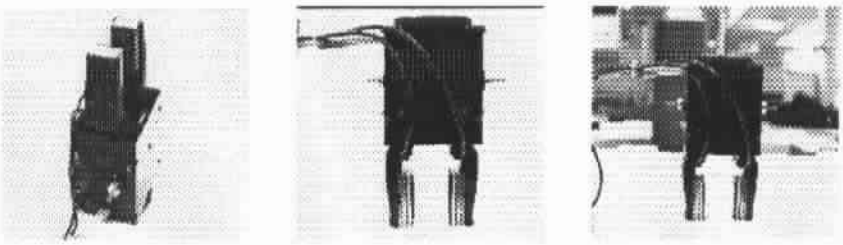


Figure 39: Different Views of the Lord Gripper

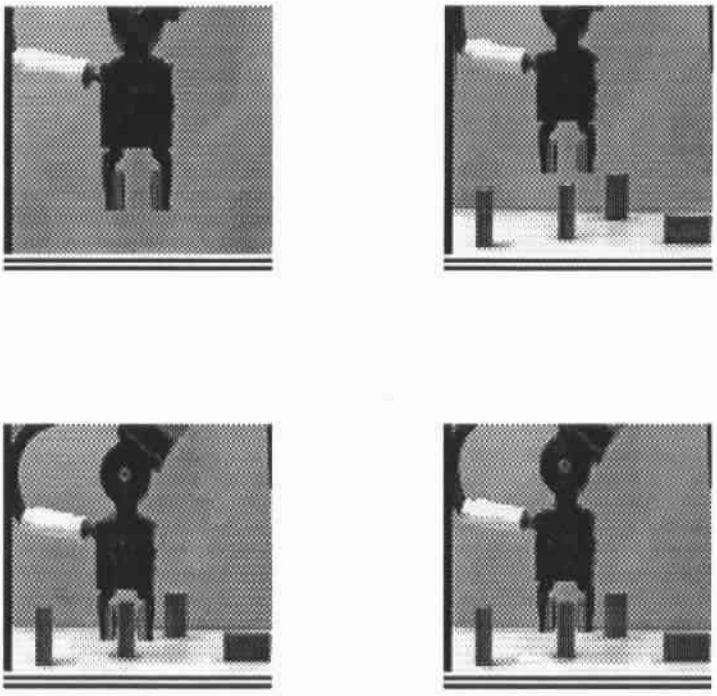


Figure 40: **A Grasping Task :** As seen by the observer's camera

track of the different observer states and to navigate in the observer automaton.

14.2.4 Examples

Experiments were performed to observe the robot hand. The Lord experimental gripper is used as the manipulating hand. Different views of the gripper are shown in Figure 39. Tracking is performed for some features on the gripper in real time. The visual tracking system works in real time and a position control vector is supplied to the observer manipulator.

Some visual states for a grasping task using the Lord gripper, as seen by the observer camera, is shown in figure 40. The sequence is defined by our model, and the visual states correspond to the gripper movement as it approaches an object an then grasps it.

The full system is implemented and tested for some simple visual action sequences. One such example is shown in figure 41. The automaton encodes an observer which tracks the hand by

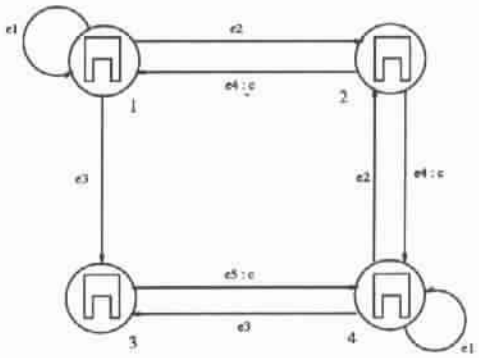


Figure 41: A Model for a Simple Visual Sequence

keeping a fixed geometric relationship between the observer's camera and the hand as so long as the hand does not approach the observer's camera rapidly. In that case, the observer tends to move sideways, that is, dodge and start viewing and tracking from sideways. It can be thought of as an action to avoid collision, due to the fact that the intersection of the workspaces of both robots is not empty. State 1 represents the visual situation where the hand is in a centered viewing position with respect to the observer and viewed from a frontal position. State 2 represents the hand in a non-centered position and tending to escape the visual view, but not approaching the observer rapidly. State 3 represents a "dangerous" situation as the hand has approached the observer rapidly. State 4 represents the hand being viewed from sideways, and the hand is centered within the imaging plane.

After having defined the states, the events causing state transitions can be easily described. Event e_1 represents no hand movements, event e_2 represents all hand movements in which the hand does not approach the camera rapidly. Event e_3 represents a large movement towards the observer. Events e_4 and e_5 are controllable tracking events, where e_4 always compensates for e_2 in order to keep a fixed 3-D relationship and e_5 is the "dodging" action where the observer moves to start viewing from sideways, while keeping the hand in a centered position.

The events can thus be defined precisely as ranges on the recovered world motion parameters. For example, e_3 can be defined as any motion $V_Z \geq d_z$. Event e_1 is defined as any motion such that :

$$-\epsilon_x \leq V_X \leq \epsilon_x \wedge -\epsilon_y \leq V_Y \leq \epsilon_y \wedge -\epsilon_z \leq V_Z \leq \epsilon_z$$

It should be noted that defining e_1 in this manner helps a lot in suppressing noise. Having defined the events, the task reduces to computing the relevant areas under the distribution curves for the various 3-D motion parameters and computing the probabilities for the ranges of e_1, e_2 and e_3 at states 1 and 4. State transitions is asserted and reported when the probability value exceeds a preset threshold. States 1 and 4 are considered to be the set of stable states, by enabling the tracking events e_4 and e_5 the system can be made stable with respect to that set.

The low level visual feature acquisition is performed on the MaxVideo pipelined video processor at frame rate. The state machine resides on a Sun SparcStation 1. The Lord gripper is mounted

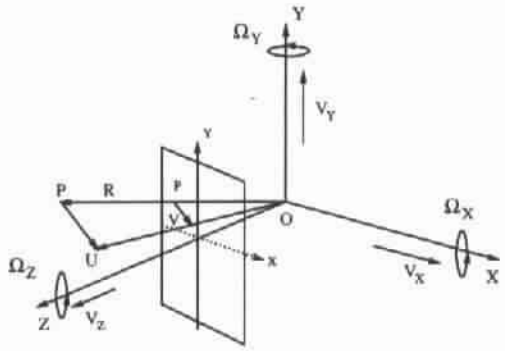


Figure 42: 3-D Formulation for Stationary Scene/Moving Viewer

on a PUMA 560 arm and the observer's camera is mounted on a second PUMA 560.

14.3 Identifying Motion Events

We use the image motion to estimate the hand movement. This task can be accomplished by either feature tracking or by computing the full optic flow. The image flow detection technique we use is based on the sum-of-squared-differences optic flow. The sensor acquisition procedure (grabbing images) and uncertainty in image processing mechanisms for determining features are factors that should be taken into consideration when we compute the uncertainty in the optic flow.

One can model an arbitrary 3-D motion in terms of stationary-scene/moving-viewer as shown in Figure 42. The optical flow at the image plane can be related to the 3-D world as indicated by the following pair of equations for each point (x, y) in the image plane [134] :

$$v_x = \left\{ x\frac{V_Z}{Z} - \frac{V_X}{Z} \right\} + \left[xy\Omega_X - \left(1 + x^2\right)\Omega_Y + y\Omega_Z \right]$$

$$v_y = \left\{ y\frac{V_Z}{Z} - \frac{V_Y}{Z} \right\} + \left[\left(1 + y^2\right)\Omega_X - xy\Omega_Y - x\Omega_Z \right]$$

where v_x and v_y are the image velocity at image location (x, y), (V_X, V_Y, V_Z) and $(\Omega_X, \Omega_Y, \Omega_Z)$ are the translational and rotational velocity vectors of the observer, and Z is the unknown distance from the camera to the object. In this system of equations, the only knowns are the 2-D vectors v_x and v_y, if we use the formulation with uncertainty then basically the 2-D vectors are random variables with a known probability distribution. A number of techniques can be used to linearize the system of equations and to solve for the motion and structure parameters as random variables [14,15,190].

14.4 Modeling and Recovering 3-D Uncertainties

The uncertainty in the recovered image flow values results from sensor uncertainties and noise and from the image processing techniques used to extract and track features. We use a static camera

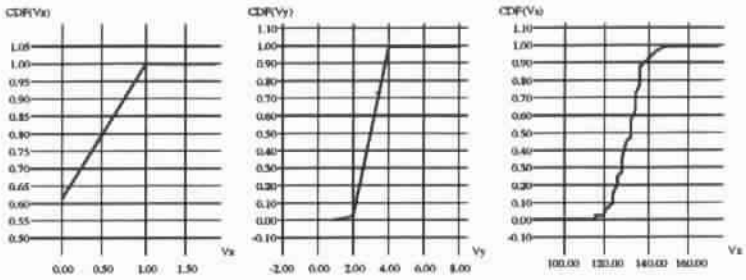
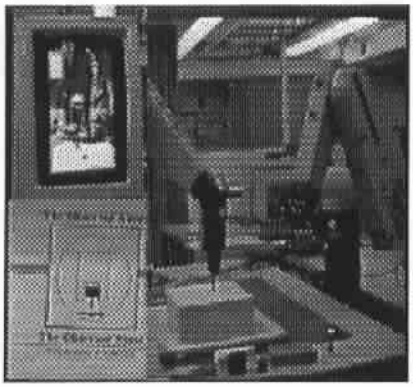


Figure 43: Cumulative Density Functions of the Translational Velocity

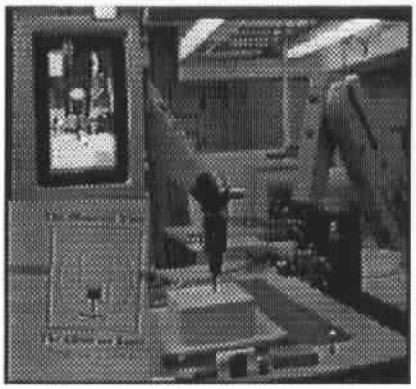
calibration technique to model the uncertainty in 3-D to 2-D feature locations. The strategy used to find the 2-D uncertainty in the features 2-D representation is to utilize the recovered camera parameters and the 3-D world coordinates (x_w, y_w, z_w) of a known set of points and compute the corresponding pixel coordinates, for points distributed throughout the image plane a number of times, find the actual feature pixel coordinates and construct 2-D histograms for the displacements from the recovered coordinates for the experiments performed. The number of the experiments giving a certain displacement error would be the z axis of this histogram, while the x and y axis are the displacement error. The three dimensional histogram functions are then normalized such that the volume under the histogram is equal to 1 unit volume and the resulting normalized function is used as the distribution of pixel displacement error.

The spatial uncertainty in the image processing technique can be modeled by using synthesized images and corrupting them, then applying the feature extraction mechanism to both images and computing the resulting spatial histogram for the error in finding features. The probability density function for the error in finding the flow vectors can thus be computed as a spatial convolution of the sensor and strategy uncertainties. We then eliminate the unrealistic motion estimates by using the physical (geometric and mechanical) limitations of the manipulating hand. Assuming that feature points lie on a planar surface on the hand, then we can develop bounds on the coefficients of the motion equations, which are second degree functions in x and y in three dimensions, $v_x = f_1(x, y)$ and $v_y = f_2(x, y)$.

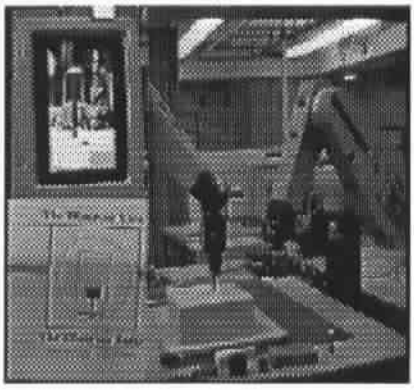
The 2-D uncertainties are then used to recover the 3-D uncertainties in the motion and structure parameters. The system is linearized by either dividing the parameter space into three subspaces for the translational, rotational and structure parameters and solving iteratively or using other linearization techniques and/or assumptions to solve a linear system of random variables [14,15,21,190,194,198]. As an example, the recovered 3-D translational velocity cumulative density functions for an actual world motion, $V_X = 0\ cm$, $V_Y = 0\ cm$ and $V_Z = 13\ cm$, is shown in figure 43. It should be noted that the recovered distributions represents a fairly accurate estimation of the actual 3-D motion.



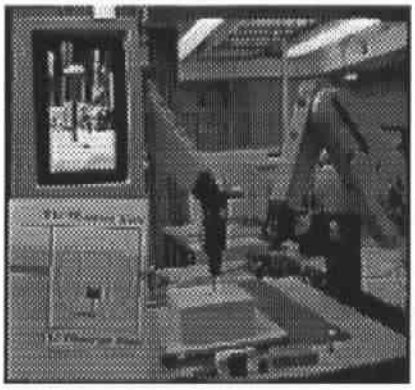
(1)



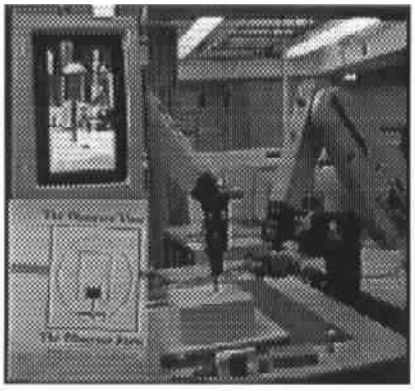
(2)



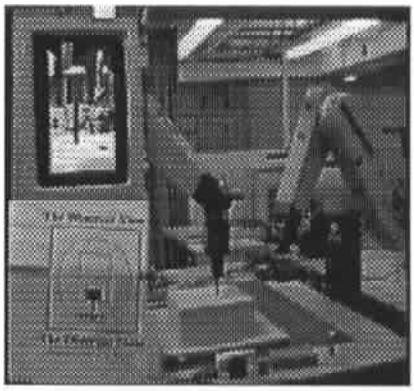
(3)



(4)

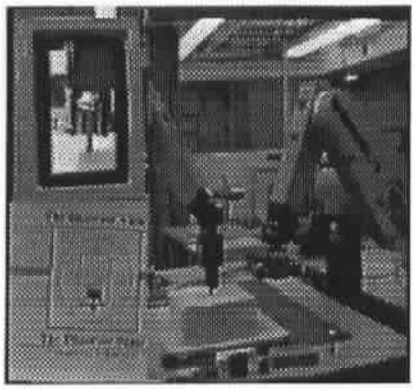


(5)

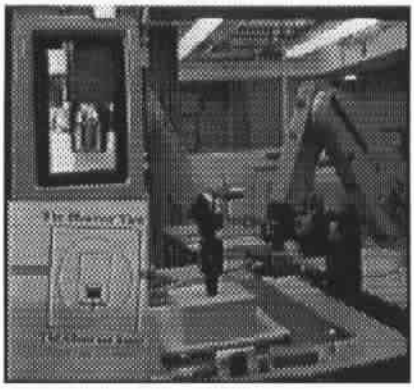


(6)

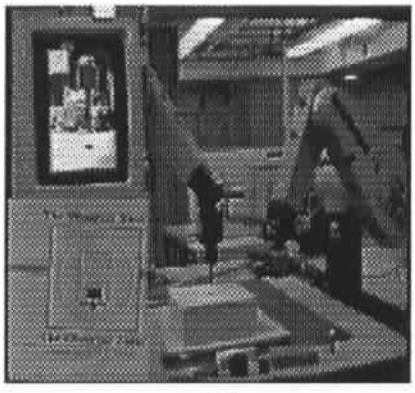
Figure 44: Observer State and View (1)



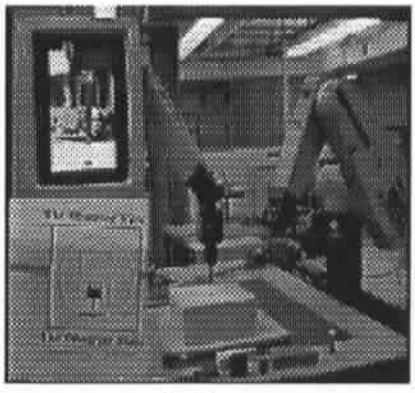
(7)



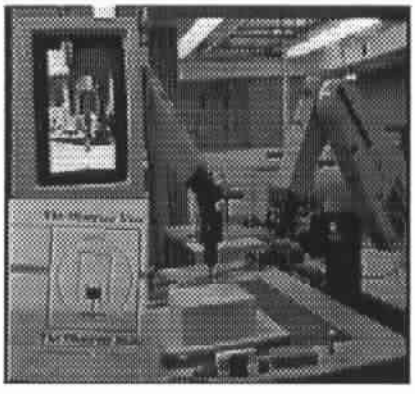
(8)



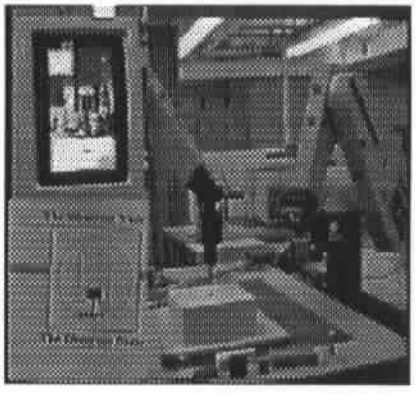
(9)



(10)



(11)



(12)

Figure 45: Observer State and View (2)

Thus, we have proposed a new approach to solving the problem of observing a moving agent. Our approach uses the formulation of discrete event dynamic systems as a high-level model for the framework of evolution of the visual relationship over time. The proposed formulation can be extended to accommodate for more manipulation processes. Increasing the number of states and expanding the events set would allow for a variety of manipulating actions.

15 Sensing for Inspection of Machine Parts

This work addresses the application of discrete event dynamic systems (DEDS) for autonomous sensing and inspection as part of the reverse engineering process. A dynamic recursive context for DEDS is presented and its usage for managing a complex hybrid system which has continuous, discrete and symbolic aspects is illustrated. We suggest that the dynamic recursive context is aptly suited to controlling and observing the active inspection of machined parts using such a hybrid system.

Reverse engineering is the process of constructing an accurate representation from sensed data. It can be represented by a closed loop system that consists of four main modules:

- Sensing
- CAD Modeling
- Manufacturing
- Inspection

This closed loop system is the framework we used to develop an integrated CAD/CAM/sensing system for inspection and reverse engineering. The process starts by constructing an initial CAD model using 2-d and 3-d vision, then the inspection module uses this model to drive a coordinate measuring machine (CMM). The results are used to increase the accuracy of the model. Additional sensing iterations could be made until the desired accuracy is obtained. Figure 46 shows this closed loop system.

Most research in reverse engineering ([172, 141, 99, 100, 101, 49, 50]) concentrates on the sensing and fitting techniques required. Hsieh[103] describes a system which does sculptured surface reconstruction with a CMM. The focus of the work is on path planning and surface fitting. If errors occur while gathering data, the system aborts and must be restarted. Van Thiel [200] describes an interactive CMM inspection system. The user is included as part of the control loop, and can abort inspections and call for explorations of particular features. Our work describes an approach that automatically gathers the sense data, processes it, and makes decisions based upon it for reverse engineering.

We use a recursive dynamic strategy for exploring machine parts. A discrete event dynamic system (DEDS) framework is designed for modeling and structuring the sensing and control problems. The dynamic recursive context for finite state machines (DRFSM) is a DEDS representation tailored to the recursive nature of the mechanical parts under consideration.

DRFSM is particularly useful for controlling the inspection module, and this has been an important aspect of our research.

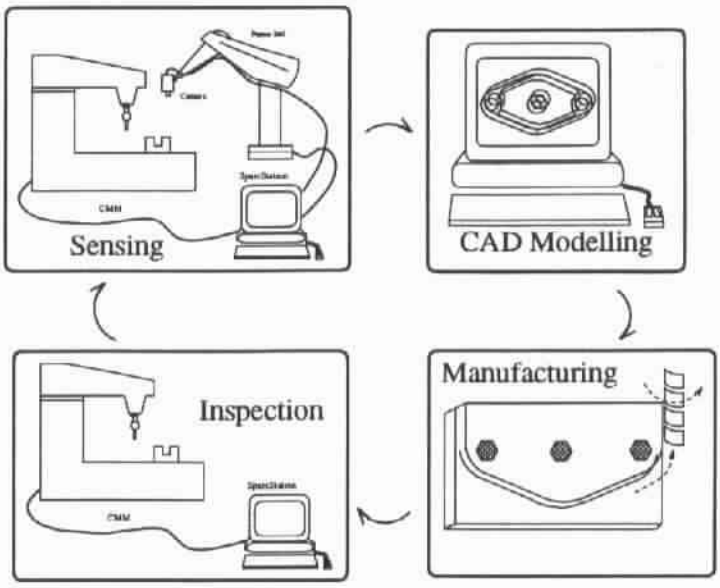


Figure 46: Closed loop system for reverse engineering

15.1 Modeling and Constructing an Observer

A DEDS framework is used to model the tasks that the autonomous observer system executes. This model is used as a high level structuring technique to preserve and make use of the information we know about the way in which a mechanical part should be explored. The state and event description is associated with different visual cues; for example, appearance of objects, specific 3-D movements and structures, interaction between the touching probe and part, and occlusions. A DEDS observer serves as an intelligent sensing module that utilizes existing information about the tasks and the environment to make informed tracking and correction movements and autonomous decisions regarding the state of the system.

To be able to determine the current state of the system we need to observe the sequence of events occurring in the system and make decisions regarding the state of the automaton. State ambiguities are allowed to occur, however, they are required to be resolvable after a bounded interval of events. In a *strongly output stabilizable* system, the state of the system is known at bounded intervals and allowable events can be controlled (enabled or disabled) in a way that ensures return in a bounded interval to one of a desired and known set of states.

One of the objectives is to make the system strongly output stabilizable and/or construct an observer to satisfy specific task-oriented visual requirements. Many 2-D visual cues for estimating 3-D world behavior can be used. Examples include: image motion, shadows, color and boundary information. The uncertainty in the sensor acquisition procedure and in the image processing mechanisms are taken into consideration to compute the world uncertainty.

15.2 Experiments

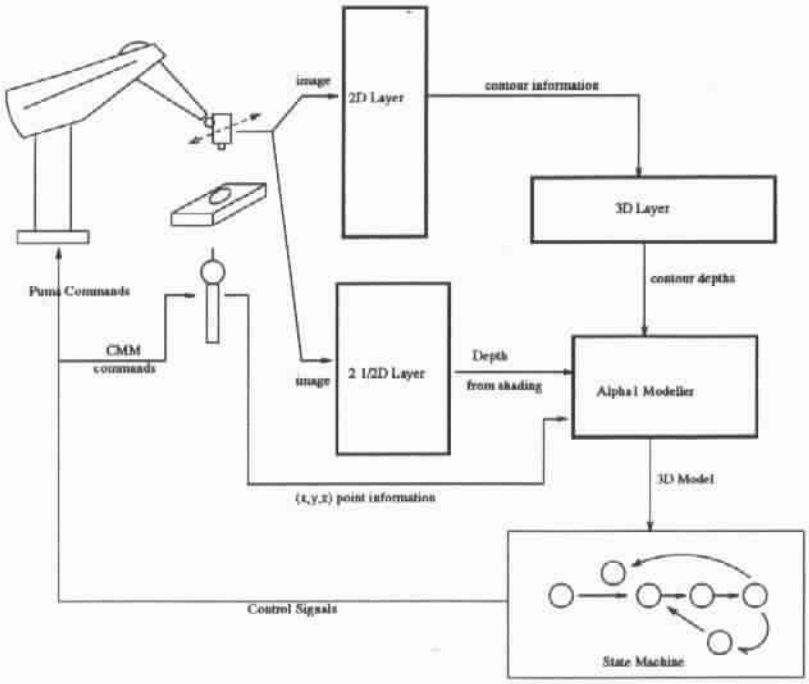


Figure 47: Inspection system overview

In conducting our experiments, we use a B/W CCD camera mounted on a Puma 560 robot arm, that observe and guide the interaction between the CMM probe and the machined part (see Figure 47.) In order for the state machine to provide control, it must be aware of state changes in the system. As inspection takes place, the camera supplies images that are interpreted by a set of 2D and 3D vision processing algorithms and used to drive the DRFSM. These algorithms are described in greater detail in other publications [188, 179, 181, 182, 180, 184], but include thresholding, edge detection, region growing, stereo vision, etc. The robot arm is used to position the camera in the workplace and move in the case of occlusion problems.

The object of these experiments was to test the operation of the visual system with the state machine. Two facets of this were the generation of an initial model from stereo vision and the generation of events that describe a probe's relationship to features in that model.

This stereo process used the Puma arm to gather pairs of images. The resulting model was used to determine feature relationships used in the DEDS controller. The models shown are from this initial visual inspection.

The event generation method, consisting of 2-d image processing routines, was used to detect the relationship of a simulated (hand-held) CMM probe to the features in the initial model. These events were processed by the controller, which output text messages guiding the experimenter to move the probe or indicate that a touch had occurred.

The automaton used in the environment is shown in Figure 48. This machine has the following states:

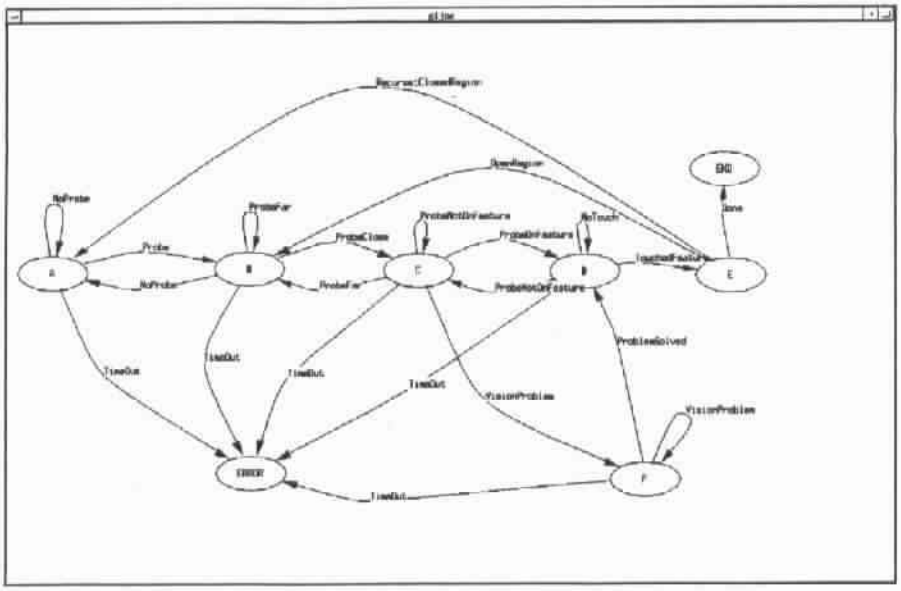


Figure 48: Inspection Environment Window

- **A:** The initial state, waiting for the probe to appear.
- **B:** The probe appears, and waiting for it to be close. Here, "close" is a measure of the distance between the probe and the current feature, since it depends on the level of the recursive structure. For example, the distance at the first level, which represents the outer contours or features, is larger than that of the lower levels.
- **C:** Probe is close, but not on feature.
- **D:** The probe appears to be on feature in the image, and waiting for physical touch indicated from the CMM machine.
- **E:** Physical touch has happened (and the CMM measurements for the feature parameters are recorded and saved for updating the CAD model.) If the current feature represents a closed region, the machine goes one level deeper to get the inner features by a recursive call to the initial state after changing the variable transition parameters. If the current feature was an open region, then the machine finds any other features in the same level.
- **F:** This state is to solve any vision problem happens during the experiment. For example, if the probe is occluding one of the features, then the camera position can be changed to solve this problem.
- **ERROR:** There is a time limit for each part of this experiment. If for any reason, one of the modules doesn't finish in time, the machine will go to this state, which will report the error and terminate the experiment.

Figure 49: Experimental Setup

15.2.1 Experimental results, Automated Bracket Inspection

A metal bracket was used in the experiment to test the inspection automaton. The piece was placed on the inspection table within view of the camera (see Figure 49).

The machine was brought on line and execution begun in State A, the start state. After initiating the inspection process, the DRFSM transitioned through states until the probe reached the bracket boundary. The state machine then called for the closed region to be recursively inspected until finally, the hole was explored and the machine exited cleanly. The sequence is shown in Figure 52.

The original part and the resulting reverse-engineered part are shown in Figures 50 (wire-frames) and 51 (rendered images). Notice that the two side holes and a portion of the bracket were not sensed correctly, as a simple strategy was used to sense from only one direction. In the next experiment, a more complicated model is sensed with a more sophisticated sensing and modeling strategy.

15.2.2 Experimental Results, Cover Plate

A second experiment was run in a similar fashion, using a part similar to the fuel pump cover from a Chevrolet engine. This piece offers interesting features and has a complex recursive structure which allowed us to test the recursive nature of the state machine.

The sensing strategy used here was more robust than in the previous experiment. Detected feature contours were sensed with stereo vision and used to build up a feature-based α_1 model. This model was then used to semi-automatically machine a reproduction of the part. The original

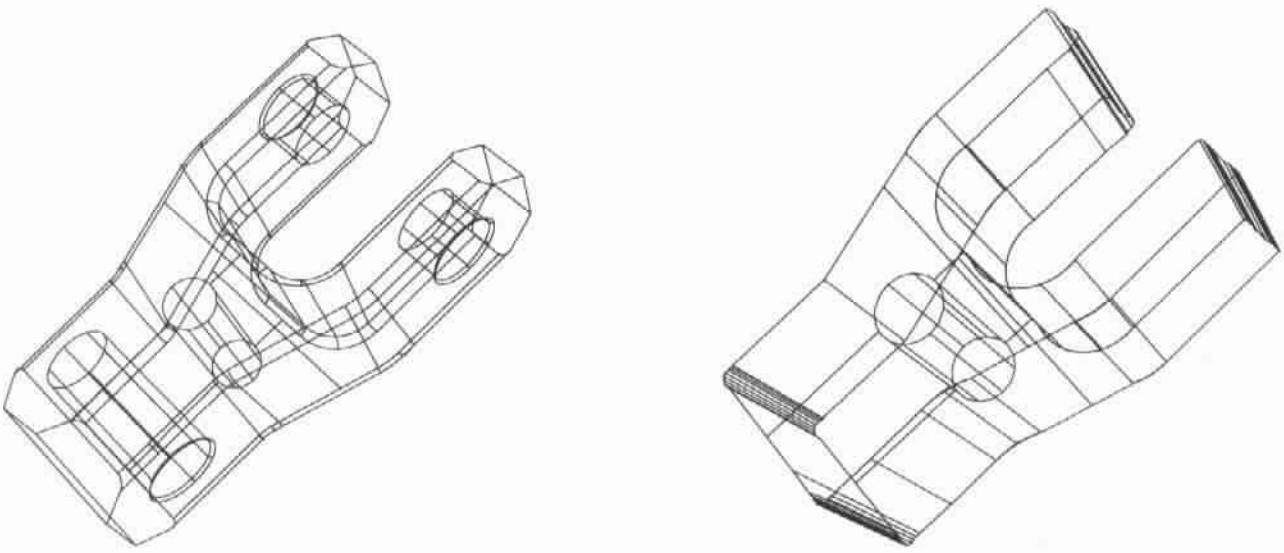


Figure 50: Original and Reverse-Engineered part models

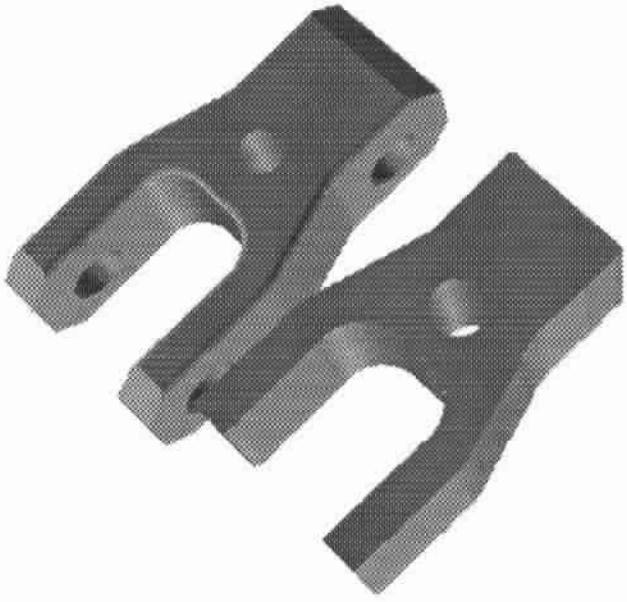
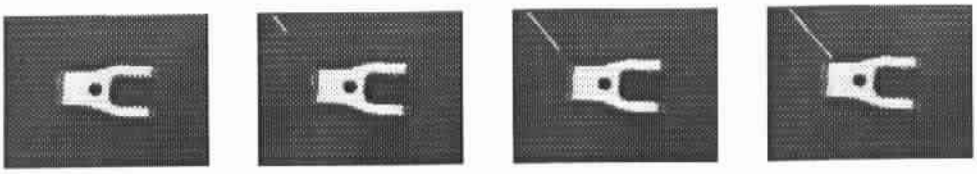
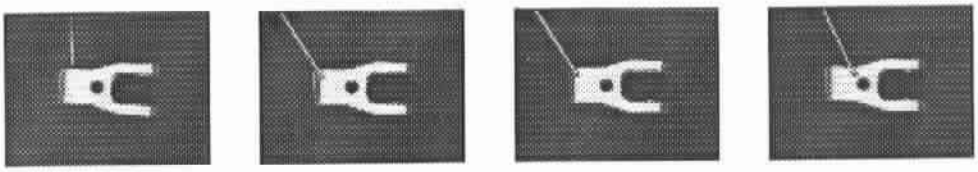


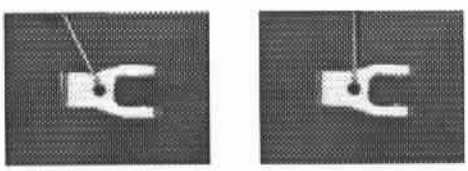
Figure 51: Original and reproduction



State A: NoProbe State B: ProbeFar State C: ProbeClose State D: ProbeOnFeature



State E: TouchedFeature State A: NoProbe State B: ProbeFar State C: ProbeClose



State D: ProbeOnFeature State E: TouchedFeature

Figure 52: Bracket Sequence

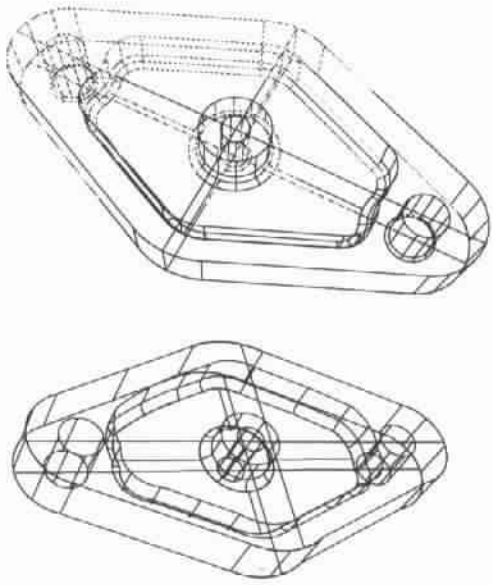


Figure 53: Original and Vision-Reverse Engineered Models

and reverse-engineered wireframe models are shown in Figures 53. A photograph of the original and reproduction is shown in 54. For more detail on the sensing strategy, please see [188].

The inspection sequence corresponding to this experiment is shown in Figure 55. Shown there, the DRFSM transitions correctly through the inspection of the outside profile (depth of recursion=0), a hole (1), a profile pocket (1), a hole (2), and another hole (1).

16 Conclusions

We have reviewed the Discrete Event Systems area, presented several frameworks used in DES modeling, and discussed the mathematical basis for some of them. Some evaluation criteria for these frameworks were also discussed.

A software environment system was developed for simulating, analyzing, synthesizing, monitoring, and controlling complex discrete event and hybrid systems. We have also presented two problems related to robotics and automation for which discrete event and hybrid systems formulation play a significant role in the solution.

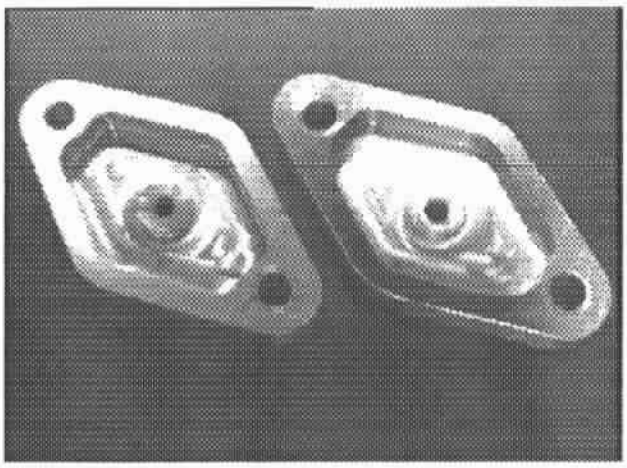


Figure 54: Original and Vision-Reverse Engineered Parts

References

[1] T. Agerwala, "Putting Petri nets to work," in *Modeling and Control of Automated Manufacturing Systems* (A. A. Desrochers, ed.), Washington, D. C.: IEEE Computer Society Press, 1990.

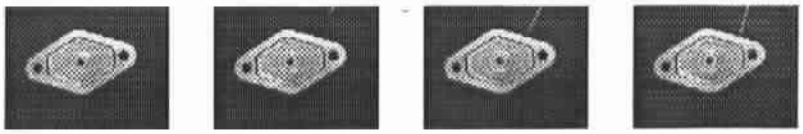
[2] R. Akella, O. Maimon, and S. Gershwin, "Value function approximation via linear programming for FMS scheduling," *International Journal of Production Research*, vol. 28, no. 8, pp. 1459–1470, 1990.

[3] R. Y. Al-Jaar and A. A. Desrochers, "A survey of Petri nets in flexible manufacturing systems," in *Proceedings of 1988 IMACS Conf.*, July 1988.

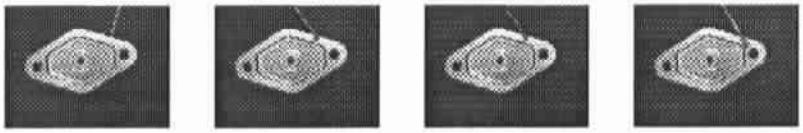
[4] R. Y. Al-Jaar and A. A. Desrochers, *Advances in automation and robotics*, vol. 2, ch. Petri nets in automation and manufacturing, pp. 153–225. JAI Press, 1990.

[5] H. Alla, P. Ladet, J. Martinez, and M. Silva, "Modeling and validation of complex systems by colored Petri nets application to a flexible manufacturing system," in *Advances in Petri Nets 1984*, vol. 188 of *Lecture Notes in Computer Science*, pp. 15–31, Berlin: Springer–Verlag, 1984.

[6] J. Aloimonos and A. Bandyopadhyay, "Active Vision". In *Proceedings of the* 1^{st} *International Conference on Computer Vision*, 1987.

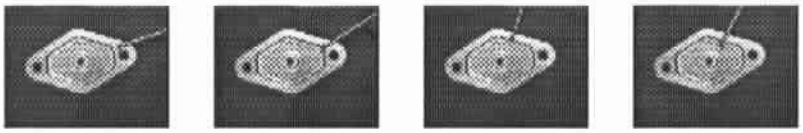


State A: NoProbe State B: ProbeFar State C: ProbeClose State D: ProbeOnFeature

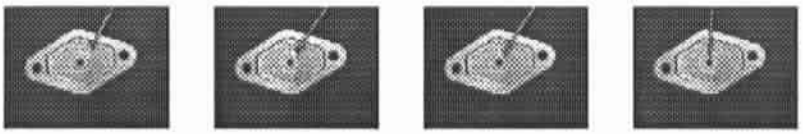


State E: TouchedFeature State A: NoProbe State B: ProbeFar State C: ProbeClose

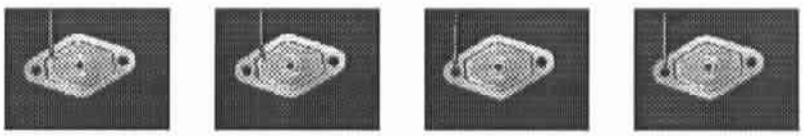
State D: ProbeOnFeature State E: TouchedFeature State A: NoProbe State B: ProbeFar



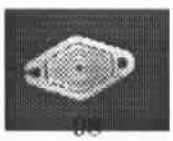
State C: ProbeClose State D: ProbeOnFeature State E: TouchedFeature State A: NoProbe



State B: ProbeFar State C: ProbeClose State D: ProbeOnFeature State E: TouchedFeature



State A: NoProbe State B: ProbeFar State C: ProbeClose State D: ProbeOnFeature



State E: TouchedFeature

[7] P. Anandan, "A Unified Perspective on Computational Techniques for the Measurement of Visual Motion". In *Proceedings of the 1st International Conference on Computer Vision*, 1987.

[8] P. J. Antsaklis and K. M. Passino, eds., *An Introduction to Intelligent and Autonomous Control.* Boston: Kluwer Academic Publishers, 1993.

[9] F. Baccelli, G. Cohen, G. J. Olsder, and J. P. Quadrat, *Algebraic and stochastic analysis of timed discrete event systems.* Wiley, 1991.

[10] F. Baccelli, G. Cohen, G. J. Olsder, and J. P. Quadrat, *Synchronization and Linearity.* Wiley, 1992.

[11] F. Baccelli and Z. Liu, "On the execution of parallel computations on multiprocessor systems—a queueing theory approach," *Journal of the ACM*, vol. 27, no. 32, pp. 373–414, 1990.

[12] Bail, J. L., Alla, H., and David, R. Hybrid petri nets. In *European Control Conference* (Grenoble, France, 1991).

[13] R. Bajcsy, "Active Perception", *Proceedings of the IEEE*, Vol. 76, No. 8, August 1988.

[14] R. Bajcsy and T. M. Sobh, *Observing a Moving Agent.* Technical Report MS-CIS-91-01 and GRASP Lab. TR 247, Computer Science Dept., School of Engineering and Applied Science, University of Pennsylvania, January 1991.

[15] Bajcsy, R., and Sobh, T. A framework for observing a manipulation process. Grasp Lab 216, University of Pennsylvania, Philadelphia, PA, June 1990. Also MS-CIS-90-34.

[16] A. D. Baker, T. L. Johnson, D. I. Kerpelman, and H. A. Sutherland, "GRAFCET and SFC as factory automation standards," in *Proceedings of 1987 American Control Conference*, (Minneapolis, MN), pp. 1725–1730, June 1987.

[17] K. Baker, *Introduction to sequencing and scheduling.* Wiley, 1974.

[18] Balemi, S. *Control of Discrete Event Systems: Theory and Application.* PhD thesis, Automatic Control Laboratory, Swiss Federal Institute of Technology (ETH), Zurich, Switzerland, May 1992.

[19] A. Ballakur and H. Steudel, "Integration of job shop control systems: A state–of–the–art review," *Journal of Manufacturing Systems*, vol. 3, no. 1, pp. 71–80, 1984.

[20] J. Banks and J. S. Carson, *Discrete–Event System Simulation.* Englewood Cliffs, NJ: Prentice–Hall, 1984.

[21] J. L. Barron, A. D. Jepson and J. K. Tsotsos, "The Feasibility of Motion and Structure from Noisy Time-Varying Image Velocity Information", *International Journal of Computer Vision*, December 1990.

[22] D. D. Bedworth, M. R. Henderson, and P. M. Wolfe, *Computer–Integrated Design and Manufacturing.* McGraw–Hill, 1991.

[23] R. Bellman and S. Dreyfus, *Applied dynamic programming.* Princeton, NJ: Princeton Univ. Press, 1962.

[24] M. Ben-Ari, *Principles of concurrent programming.* Prentice–Hall, 1982.

[25] S. Bennett, *Real-time computer control: an introduction.* Englewood Cliffs, NJ: Prentice–Hall, 1988.

[26] A. Benveniste, P. Le Guernic, and C. Jacquemot, "The SIGNAL software environment for real-time system specification, design, and implementation," in *Proceedings of 1989 IEEE Work. CACSD*, Dec. 1989.

[27] J. Billingsley, *Controlling with computers: control theory and practical digital systems.* McGraw–Hill, 1989.

[28] J. T. Black, *The Design of the Factory with a Future.* McGraw–Hill, 1991.

[29] G. V. Bochmann, *Distributed System Design.* Springer–Verlag, 1983.

[30] P. Bratley, B. Fox, and L. Schrage, *A Guide to Simulation.* New York: Springer–Verlag, 2nd ed., 1987.

[31] Y. Brave and M. Heymann, "Control of discrete event systems modeled as hierarchical state machines," Tech. Rep. #9012, Center for Intelligent Systems, Technion – Israel Institute of Technology, Haifa, Israel, Mar. 1991.

[32] P. Brémaud, *Point Processes and Queues–A Martingale approach.* Springer–Verlag Series in Statistics, New York: Springer–Verlag, 1981.

[33] W. L. Brogan, *Modern Control Theory.* Englewood Cliffs, NJ: Prentice–Hall, 3rd ed., 1991.

[34] S. D. Brookes, C. A. R. Hoare, and A. W. Roscoe, "A theory of communicating sequential processes," *Journal of the ACM*, vol. 31, no. 3, pp. 560–599, 1984.

[35] A. Burns and A. Wellings, *Real–Time Systems and Their Programming Languages.* Addison–Wesley, 1990.

[36] P. J. Burt, et al., "Object Tracking with a Moving Camera", *IEEE Workshop on Visual Motion*, March 1989.

[37] J. A. Buzacott and D. D. Yao, "Flexible manufacturing systems: a review of analytical models." Management Science.

[38] T. Cao and A. C. Sanderson, "Task decomposition and analysis of assembly sequence plans using Petri nets," in *Proceedings of 3rd Int. Conf. CIM*, pp. 138–147, May 1992.

[39] X.-R. Cao, "On a sample performance function of Jackson queueing networks," *Operations Research*, vol. 36, pp. 128–136, 1988.

[40] Xi-Ren Cao, "The Predictability of Discrete Event Systems", *Proceedings of the* 27^{th} *Conference on Decision and Control*, December 1988.

[41] X.-R. Cao and Y. Dallery, "An operational approach to perturbation analysis of closed queueing networks," in *Proceedings of 1986 American Control Conference*, 1986.

[42] C. G. Cassandras, "Optimizing recirculation in flexible manufacturing systems," in *Proceedings of 2nd ORSA/TIMS Conf. Flexible Manufacturing Systems: Operations Research Models and Applications* (K. E. Stecke and R. Suri, eds.), pp. 381–392, Elsevier, 1986.

[43] C. G. Cassandras, "On the duality between routing and scheduling systems with finite buffer space," in *Proceedings of 31st IEEE Conference on Decision and Control*, pp. 2364–2365, Dec. 1992.

[44] C. G. Cassandras, *Discrete Event Systems: Modeling and Performance Analysis.* The Aksen Associates Series in Electrical and Computer Engineering, Homewood, IL 60430: Aksen Associates Incorporated Publishers, 1993.

[45] C. G. Cassandras and J. I. Lee, "Applications of perturbation techniques to optimal resource sharing in discrete event systems," in *Proceedings of 1988 American Control Conference*, (Atlanta, GA), pp. 450–455, 1988.

[46] U. Chandrasekaran and S. Sheppard, "Discrete event distributed simulation—a survey," in *Proceedings of Conference on Methodology and Validation*, pp. 32–37, 1987.

[47] P. R. Chang, *Parallel Algorithms and VLSI architectures for robotics and assembly scheduling.* PhD thesis, Purdue Univ., West Lafayette, IN, Dec. 1988.

[48] F. Chaumette and P. Rives, "Vision-Based-Control for Robotic Tasks", In *Proceedings of the IEEE International Workshop on Intelligent Motion Control*, Vol. 2, pp. 395-400, August 1990.

[49] CHEN, Y., AND MEDIONI, G. Object modelling by registration of multiple range images. *International Journal of Image and Vision Computing 10*, 3 (Apr. 1992), 145–155.

[50] CHEN, Y., AND MEDIONI, G. Integrating multiple range images using triangulation. In *Image Understanding Workshop* (April 1993), Defense Advanced Research Projects Agency, Software and Intelligent Systems Office, pp. 951–958.

[51] E. K. P. Chong and P. J. Ramadge, "Convergence of recursive optimization algorithms using IPA derivative estimates," in *Proceedings of 1990 American Control Conference*, (San Diego, CA), May 1990.

[52] COLOMBO, A. W., MARTINEZ, J., AND CARELLI, R. Formal validation of complex production systems using coloured petri nets. In *Proc. 1994 IEEE Int. Conf. Robotics and Automation* (Can Diego, CA, USA, May 1994), IEEE, pp. 1713–1718.

[53] S. Connolly, Y. Dallery, and S. B. Gershwin, "A real–time policy for performing setup changes in a manufacturing system," in *Proceedings of 31st IEEE Conference on Decision and Control*, pp. 764–770, Dec. 1992.

[54] A. E. Conway and N. D. Georganas, *Queueing Networks–Exact Computational Algorithms: A Unified Theory Based on Decomposition and Aggregation.* Cambridge, MA: The MIT Press, 1989.

[55] B. Darakanada, "Simulation of manufacturing process under a hierarchical control algorithm," Master's thesis, MIT, May 1989.

[56] DAVID, R., AND ALLA, H. *Petri Nets and Grafcet.* Hermes Pub, Paris, France, 1989.

[57] R. David and H. Alla, *Petri Nets and Grafcet: Tools for modelling discrete event systems.* New York: Prentice–Hall, 1992.

[58] E. V. Denardo, *Dynamic Programming: Models and Applications.* Englewood Cliffs, NJ: Prentice–Hall, 1982.

[59] M. J. Denham and A. J. Laub, eds., *Advanced Computing Concepts and Techniques in Control Engineering*, vol. 47 of *Computer and Systems Sciences.* Springer–Verlag, 1988.

[60] P. Deransart, M. Jourdan, and B. Lohro, "A survey of attribute grammers part 1: Main result on attribute grammars," Tech. Rep. 485, INRIA, Jan. 1986.

[61] A. A. Desrochers, *Modeling and Control of Automated Manufacturing Systems.* New York: IEEE Computer Society Press, 1990.

[62] R. L. Devaney, *An Introduction to Chaotic Dynamical Systems.* Menlo Park, California: The Benjamin/Cummings Publishing Company, Inc., 1986.

[63] D. Dilts, N. Boyd, and H. Whorms, "The evolution of control architectures for automated manufacturing," *Journal of Manufacturing Systems*, vol. 10, no. 1, pp. 79–93, 1991.

[64] B. R. Donald, *Error Detection and Recovery in Robotics*, vol. 336 of *Lecture Notes in Computer Science.* Berlin: Springer–Verlag, 1989.

[65] S. Drees *et al.*, *Advances in Petri Nets 1987*, ch. Bibliography of Petri nets, pp. 309–341. Springer–Verlag, 1987.

[66] D. Dubois and K. E. Stecke, "Using Petri nets to represent production processes," in *Proceedings of 22nd IEEE Conference on Decision and Control*, (San Antonio, TX), pp. 1062–1067, Dec. 1983.

[67] C. Dupont-Gatelmand, "A survey of flexible manufacturing systems," *Journal of Manufacturing Systems*, vol. 1, no. 1, pp. 1–16, 1982.

[68] S. E. Elmaghraby, "The economic lot scheduling problem (ELSP): Review and extensions," *Management Science*, vol. 24, pp. 587–598, 1978.

[69] M. Elzas, R. I. Oren, and B. Zeigler, eds., *Modelling an simulation methodology in the artificial intelligence era.* Amsterdam: North–Holland, 1986.

[70] W. Eversheim and P. Herrmann, "Recent trends in flexible automated manufacturing," *Journal of Manufacturing Systems*, vol. 1, no. 2, pp. 139–148, 1982.

[71] A. F. Famili, D. S. Nau, and S. H. Kim, eds., *Artificial Intelligence Applications in Manufacturing.* Menlo Park, California: AAAI Press / The MIT Press, 1992.

[72] W. Feller, *An Introduction to Probability Theory and Its Applications.* New York: Wiley, 2nd ed., 1971.

[73] G. S. Fishman, *Principles of Discrete Event Simulation.* New York: John Wiley, 1987.

[74] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to NP–Completeness.* San Francisco, CA: Freeman, 1979.

[75] V. K. Garg and R. Kumar, "A state–variable approach for controlling discrete event systems with infinite states," in *Proceedings of 1992American Control Conference*, (Chicago, IL), June 1992.

[76] GHABRI, M. K., AND LADET, P. Controlled continuous petri nets. In *Proc. 1994 IEEE Int. Conf. Robotics and Automation* (Can Diego, CA, USA, May 1994), IEEE, pp. 788–793.

[77] P. Glasserman, "Derivative estimates from simulation of continuous time Markov chains," tech. rep., AT&T Bell Labs, Holmdel, NJ, 1989.

[78] P. Glasserman, *Gradient Estimation via Perturbation Analysis.* Boston: Kluwer Academic Publishers, 1991.

[79] P. Glasserman and Y. C. Ho, "Aggregation approximation for sensitivity analysis of multiclass queueing networks," *Performance Analysis*, vol. 10, pp. 295–308, 1989.

[80] M. Gondran and M. Minoux, *Graphs and algorithms.* New York: Wiley, 1986.

[81] W. B. Gong, W. Zhai, and Y. C. Ho, "Stochastic comparison algorithm for discrete optimization with Monte Carlo estimation," in *Proceedings of 31st IEEE Conference on Decision and Control*, pp. 795–802, Dec. 1992.

[82] G. Gordon, *Handbook of operations research*, ch. Simulation-computation. New York: Van Nostrand, 1978.

[83] M. A. Harrison, *Introduction to Switching and Automata Theory.* New York: McGraw–Hill, 1965.

[84] J. Hervé, P. Cucka and R. Sharma, "Qualitative Visual Control of a Robot Manipulator". In *Proceedings of the DARPA Image Understanding Workshop*, September 1990.

[85] F. S. Hiller and G. J. Lieberman, *Introduction to Operations Research.* Hoplden–Day, 4th ed., 1986.

[86] Y. C. Ho, "Adaptive design of feedback controllers for stochastic systems," *IEEE Transactions on Automatic Control*, vol. 10, pp. 367–368, July 1965.

[87] Ho, Y. Discrete event dynamical system and its application to manufacturing. In *IFAC Cong. Proc.* (1981).

[88] Y. C. Ho, "System theory and operations research – a new fusion of mathematical modeling and experimentation," in *Proceedings of Workshop on Intelligent Control 1985* (A. Saridis and A. Meystel, eds.), (Troy, NY), pp. 35–37, Aug. 1985.

[89] Y. C. Ho, *A Selected and Annotated Bibliography on Perturbation Analysis*, vol. 106 of *Lecture Notes in Control and Information Science*, pp. 217–224. Berlin: Springer–Verlag, Aug. 1987.

[90] Y. C. Ho, "Recent developments in perturbation analysis," Tech. Rep. CICS-P-209, CICS, Brown Univ., Providence, Apr. 1990.

[91] Y. C. Ho, ed., *Discrete Event Dynamic Systems: Analyzing Complexity and Performance in the Modern World.* New York: IEEE Press, 1991.

[92] Y. C. Ho, "Hierarchical production controls in a stochastic two–machine flowshop with a finite internal buffer," in *Proceedings of 31st IEEE Conference on Decision and Control*, pp. 2068–2073, Dec. 1992.

[93] Y. C. Ho, "Performance Evaluation and Perturbation Analysis of Discrete Event Dynamic Systems", *IEEE Transactions on Automatic Control*, July 1987.

[94] Y. C. Ho and X.-R. Cao, *Perturbation Analysis of Discrete Event Dynamic Systems.* Boston: Kluwer Academic Publishers, 1991.

[95] Y. C. Ho and J. Q. Hu, "An infinitesimal perturbation analysis algorithm for a multiclass G/G/1 queue," *Operations Research Letters*, vol. 9, pp. 35–44, 1990.

[96] C. A. R. Hoare, *Communicating Sequential Processes.* International Series in Computer Science, Englewood Cliffs, NJ: Prentice–Hall International, 1985.

[97] G. Hoffmann, *Discrete Event System Theory Applied to Manufacturing.* PhD thesis, Stanford, Information Systems Laboratory, Stanford University, CA, Dec. 1992.

[98] J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Languages and Computation.* Addison–Wesley Series in Computer Science, Reading, Massachusetts: Addison–Wesley, 1979.

[99] HOPPE, H., DEROSE, T., DUCHAMP, T., MCDONALD, J., AND STUETZLE, W. Surface reconstruction from unorganized points. In *Computer Graphics, SIGGRAPH '92* (July 1992), vol. 26.

[100] HOPPE, H., DEROSE, T., DUCHAMP, T., MCDONALD, J., AND STUETZLE, W. Mesh optimization. In *Computer Graphics, SIGGRAPH '93* (Aug. 1993), vol. 27.

[101] HOPPE, H., DEROSE, T., DUCHAMP, T., MCDONALD, J., AND STUETZLE, W. Piecewise smooth surface reconstruction. In *Computer Graphics, SIGGRAPH '94* (1994).

[102] B. K. P. Horn and B. G. Schunck, "Determining Optical Flow", *Artificial Intelligence*, vol. 17, 1981, pp. 185-203.

[103] HSIEH, Y. C. Reconstruction of sculptured surfaces using coordinate measuring machines. Master's thesis, Mechanical Engineering Department, University of Utah, June 1993.

[104] R. Hull and R. King, "Semantic database modeling: survey, applications, and research issues," *ACM Computing Surveys*, vol. 19, no. 3, pp. 201–260, 1987.

[105] INAN, K., AND VARAIYA, P. Finitely recursive process models for discrete event systems. *IEEE Trans. Autom. Control 33*, 7 (July 1988), 626–639.

[106] K. Jensen, *Coloured Petri Nets: A High Level Language for System Design and Analysis.* Berlin: Springer–Verlag, 1991.

[107] K. Jensen, ed., *Application and Theory of Petri Nets 1992*, vol. 616 of *Lecture Notes in Computer Science.* Berlin: Springer–Verlag, 1992.

[108] K. Jensen and G. Rozenberg, eds., *High–level Petri Nets: Theory and Applications.* Berlin: Springer–Verlag, 1991.

[109] B. Jiang, J. Black, and R. Duraisamy, "A review of recent developments in robot metrology," *Journal of Manufacturing Systems*, vol. 7, no. 4, pp. 339–357, 1988.

[110] J.-F. Kao and J. L. Sanders, "Impact of error recovery on the productivity of a unitary assembly cell," *IEEE Transactions on Robotics and Automation*, vol. 8, no. 6, pp. 730–740, 1992.

[111] J. Keilson, *Markov chain models - rarity and exponentiality.* Springer–Verlag, 1979.

[112] L. Kleinrock, *Queueing Systems*, vol. I: Theory. New York: Wiley, 1975.

[113] H. E. Koenig, Y. Tokad, and H. K. Kesavan, *Analysis of Discrete Physical Systems.* New York: McGraw–Hill, 1967.

[114] Z. Kohavi, *Switching and Finite Automata Theory.* McGraw–Hill, 1979.

[115] P. Kozák, "Methods of discrete event systems theory in AI real–time skills," in *Proceedings of the IFIP Workshop on Dependability of Artificial Intelligence Systems DAISY_91* (G. H. Schildt and J. Retti, eds.), (Vienna, Austria), pp. 271–277, Elsevier, North–Holland, Amsterdam, May 1991.

[116] P. R. Kumar and P. Varaiya, *Stochastic Systems: Estimation, Identification, and Adaptive Control.* Englewood Cliffs, NJ: Prentice–Hall, 1986.

[117] Y. Kuniyoshi, M. Inaba, and H. Inoue, "Teaching by showing : Generating robot programs by visual observation of human performance", 20^{th} ISIR, 1989.

[118] Y. Kuniyoshi, M. Inaba, and H. Inoue, "Design and implementation of a system that generates assembly programs from visual recognition of human action sequences", IROS, 1990.

[119] A. Kusiak, ed., *Intelligent Design and Manufacturing.* New York: Wiley, 1992.

[120] S. Lafortune and E. Wong, "A state model for the concurrency control problem in database management systems," in *Proceedings of 24th IEEE Conference on Decision and Control*, (Fort Lauderdale, FL), pp. 441–442, Dec. 1985.

[121] LAVIGNON, J., AND SHOHAM, Y. Temporal automata. Dept. of Computer Science STAN-CS-90-1325, Stanford University, August 1990.

[122] A. M. Law and W. D. Kelton, *Simulation Modeling and Analysis.* McGraw–Hill series in industrial engineering and management science, New York: McGraw–Hill, 2nd ed., 1991.

[123] C. Lee, "Fuzzy logic in control systems: fuzzy logic controller — part I," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 20, no. 2, pp. 404–418, 1990.

[124] C. Lee, "Fuzzy logic in control systems: fuzzy logic controller — part II," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 20, no. 2, pp. 419–435, 1990.

[125] A. M. C. Leeming, "A comparison of some discrete event simulation languages," *Simuletter*, vol. 12, no. 1–4, pp. 9–16, 1981.

[126] J. K. Lenstra and A. H. G. R. Kan, "Scheduling theory since 1981: an annotated bibliography," Tech. Rep. 188/83, Mathematisch Centrum, Amsterdam, 1983.

[127] S.-T. Levi and A. K. Agrawala, *Real–Time System Design.* New York: McGraw–Hill, 1990.

[128] H. R. Lewis and C. H. Papadimitriou, *Elements of the Theory of Computation*, Prentice-Hall, 1981.

[129] Y. Li and W. M. Wonham, "Controllability and Observability in the State-Feedback Control of Discrete-Event Systems", *Proc.* 27^{th} *Conf. on Decision and Control*, 1988.

[130] A. L. Liestman and R. H. Campbell, "A fault-tolerant scheduling problem," *IEEE Transactions on Software Engineering*, vol. 12, no. 11, pp. 1089–1095, 1986.

[131] F. Lin, *On Controllability and Observability of Discrete Event Systems.* PhD thesis, University of Toronto, Canada, 1987.

[132] F. Lin, "Robust and adaptive supervisory control of discrete event sysatems," in *Proceedings of 1992 American Control Conference*, (Chicago, IL), pp. 2804–2808, June 1992.

[133] F. Lin and W. M. Wonham, "Decentralized supervisory control of discrete–event systems," Tech. Rep. # 8612, Systems Control Group, Depertment of Electrical Engineering, University of Toronto, Canada, 1986.

[134] H. C. Longuet-Higgins and K. Prazdny, *The interpretation of a moving Retinal Image*, Proc. Royal Society of London B, 208, 385-397.

[135] E. Lopez-Mellado and R. Alami, "A failure recovery scheme for assembly workcells," in *Proceedings of 1990 International Conference on Robotics and Automation*, pp. 702–707, 1990.

[136] J. M. Maciejowski, "Data structures and software tools for computer aided design of control systems: a survey," in *IFAC Workshop on CACSD*, 1989.

[137] G. Margirier, "Flexible automated machining in france: Results of a survey," *Journal of Manufacturing Systems*, vol. 6, no. 4, pp. 253–265, 1987.

[138] M. D. Mesarović, D. Macko, and Y. Takahara, *Theory of Hierarchical, Multilevel, Systems.* New York: Academic Press, 1970.

[139] G. Meyer, "On hybrid problems in flight control," in *Workshop on Hybrid Systems*, (Cornell Univ.), MSI, June 1991.

[140] R. Milner, *Communication and Concurrency.* New York: Prentice–Hall, 1989.

[141] MOTAVALLI, S., AND BIDANDA, B. A part image reconstruction system for reverse engineering of design modifications. *J. Manufacturing Systems 10*, 5 (1991), 383–395.

[142] T. Murata, "Petri nets: Properties, analysis, and applications," *Proceedings of the IEEE*, vol. 77, no. 4, pp. 541–580, 1989.

[143] G. J. Olsder, "Applications of the theory of discrete event systems to array processors and scheduling in public transportation," in *Proceedings of 28th IEEE Conference on Decision and Control*, Dec. 1989.

[144] J. S. Ostroff, "Real–time computer control of discrete systems modelled by extended state machines: A temporal logic approach," Tech. Rep. # 8618, Systems Control Group, Depertment of Electrical Engineering, University of Toronto, Canada, 1986.

[145] C. M. Özveren, *Analysis and Control of Discrete Event Dynamic Systems: A State Space Approach.* PhD thesis, LIDS, MIT, Cambridge, MA, 1989.

[146] Özveren, C. M., Willsky, A., and Antsaklis, P. Stability and stabilizability of discrete event dynamic systems. Laboratory of Information and Decision Systems LIDS-P-1853, MIT, Cambridge, MA, 1989.

[147] C. H. Papadimitriou, *Elements of the Theory of Computation.* Englewood Cliffs, NJ: Prentice–Hall, 1981.

[148] C. H. Papadimitriou, *The Theory of Database Concurrency Control.* Rockville, MD: Computer Science Press, 1986.

[149] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity.* Englewood Cliffs, NJ: Prentice–Hall, 1982.

[150] A. Papoulis, *Probability, Random Variables and Stochastic Processes.* New York: McGraw–Hill, 3rd ed., 1991.

[151] K. M. Passino and P. J. Antsaklis, "Event rates and aggregation in hierarchical systems," *Discrete Event Dynamic Systems: Theory and Applications*, vol. 1, no. 3, pp. 271–288, 1991.

[152] J. R. Perkins and P. R. Kumar, "Stable, distributed, real–time scheduling of flexible manufacturing/assembly/disassembly systems," *IEEE Transactions on Automatic Control*, vol. 34, no. 2, pp. 139–148, 1989.

[153] J. L. Peterson, *Petri Net Theory and the Modeling of Systems.* Englewood Cliffs, NJ: Prentice–Hall, 1981.

[154] H. Plünnecke and W. Reisig, "Bibliography of Petri nets 1990," in *Advances in Petri Nets 1991* (G. Rozenberg, ed.), vol. 524 of *Lecture Notes in Computer Science*, pp. 317–572, Berlin: Springer–Verlag, June 1991.

[155] A. Pnueli, "The temporal semantics of concurrent programs," in *Semantics of Concurrent Computations* (G. Kahn, ed.), vol. 70 of *Lecture Notes in Computer Science*, (Berlin), pp. 1–20, Springer–Verlag, 1979.

[156] A. Pnueli and E. Harel, "Applications of temporal logic to the specification and verification of reactive systems: a survey of current trends," in *Current Trends in Concurrencey*, vol. 224 of *Lecture Notes in Computer Science*, (Berlin), pp. 510–584, Springer–Verlag, 1986.

[157] P. J. Ramadge, *Control and Supervision of Discrete Event Processes.* PhD thesis, University of Toronto, Canada, 1983.

[158] P. J. Ramadge, "On the periodicity of symbolic observations of piecewise smooth discrete–time systems," *IEEE Transactions on Automatic Control*, vol. 35, no. 7, pp. 807–813, 1990.

[159] P. J. Ramadge and W. M. Wonham, "Modular Feedback Logic for Discrete Event Systems", *SIAM Journal of Control and Optimization*, September 1987.

[160] P. J. Ramadge and W. M. Wonham, "Supervision of discrete event processes," in *Proceedings of 21st IEEE Conference on Decision and Control*, (Orlando, FL), pp. 1228–1229, Dec. 1982.

[161] P. J. Ramadge and W. M. Wonham, "Supervisory Control of a Class of Discrete Event Processes", *SIAM Journal of Control and Optimization*, January 1987.

[162] R. Ravichandran and A. K. Chakravarty, "Decision support in flexible manufacturing systems using timed Petri nets," *Journal of Manufacturing Systems*, vol. 5, no. 2, pp. 89–101, 1986.

[163] W. Reisig, *A Primer in Petri Net Design.* Berlin: Springer–Verlag, 1992.

[164] G. E. Révész, *Introduction to Formal Languages*, McGraw-Hill, 1985.

[165] L. C. G. Rogers and D. Williams, *Diffusions, Markov Processes, and Martingales*, vol. 2. New York: Wiley, 1987.

[166] S. M. Ross, *Introduction to Stochastic Dynamic Programming.* New York: Academic Press, 1983.

[167] R. Rubinstein, *Monte Carlo Optimization, Simulation, and Sensitivity Analysis of Queueing Networks.* New York: Wiley, 1986.

[168] R. Rubinstein, "The score function approach of sensitivity analysis of computer simulation models," *Mathematics and Computation in Simulation*, vol. 28, pp. 351–379, 1986.

[169] S. Rudneau, *Boolean Functions and Equations.* North–Holland, 1974.

[170] A. Salomaa, *Theory of Automata.* Oxford, UK: Pergamon Press, 1969.

[171] B. Sarikaya and G. V. Bochmann, eds., *Protocol Specification, Testing, and Verification*, vol. VI. North–Holland, 1987.

[172] SARKAR, B., AND MENQ, C. Smooth-surface approximation and reverse engineering. *Computer Aided Design 23*, 9 (November 1991), 623–628.

[173] R. J. Schilling, *Fundamentals of Robotics: Analysis and Control.* Englewood Cliffs, NJ: Prentice–Hall, 1990.

[174] S. Schneider, *Correctness and communication of real-time systems.* PhD thesis, Oxford Univ., Mar. 1990.

[175] J. M. Schumacher, "Discrete events: Perspectives from system theory," *CWI Quarterly*, vol. 2, no. 2, pp. 131–146, 1989.

[176] W. K. Shih, J. W. S. Liu, and J. Y. Chung, "Fast algorithms for scheduling tasks with ready times and deadlines to minimize total error," in *Proceedings of 10th IEEE Real-Time Systems Symp.*, Dec. 1989.

[177] M. Silva and R. Valette, *Petri Nets in Flexible Manufacturing*, pp. 375–417. Advances in Petri Nets, Berlin: Springer–Verlag, 1990.

[178] J. L. Snowdon and J. C. Ammons, "A survey of queueing network packages for the analysis of manufacturing systems," *Manufacturing Review*, vol. 1, pp. 14–25, 1988.

[179] SOBH, T., JAYNES, C., DEKHIL, M., AND HENDERSON, T. *Intelligent Systems: Safety, Reliability.* Springer-Verlag, Berlin, 1993, ch. Automated Inspection and Reverse Engineering, pp. 95–122.

[180] SOBH, T. M., DEKHIL, M., JAYNES, C., AND HENDERSON, T. A perception framework for inspection and reverse engineering. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR '93)* (June 1993). New York City.

[181] SOBH, T. M., DEKHIL, M., AND OWEN, J. C. Discrete event control for inspection and reverse engineering. In *IEEE International Conference on Robotics and Automation* (May 1994). San Diego.

[182] SOBH, T. M., OWEN, J. C., DEKHIL, M., JAYNES, C., AND HENDERSON, T. Industrial inspection and reverse engineering. In *IEEE 2nd CAD-Based Vision Workshop* (February 1994). Pittsburgh.

[183] SOBH, T. Discrete event dynamic systems: An overview. GRASP LAB 264, Dept. of Computer and Information Science, University of Pennsylvania, Philadelphia, PA, May 1991. Also MS-CIS-91-39.

[184] SOBH, T. A framework for visual observation. GRASP Lab 261, University of Pennsylvania, Philadelphia, PA, May 1991. Also MS-CIS-91-36.

[185] T. Sobh, "Performance evaluation via perturbation analysis," Tech. Rep. 263, GRASP Lab, University of Pennsylvania, Philadelphia, PA, May 1991.

[186] T. Sobh, *Active Observer: A Discrete Event Dynamic System Model for Controlling an Observer Under Uncertainty.* PhD thesis, University of Pennsylvania, Philadelphia, PA, Dec. 1991.

[187] SOBH, T., BAJCSY, R., AND JAMES, J. Visual observation for hybrid intelligent control implementation. In *31st IEEE CDC* (Tucson, AZ, USA, December 1992), IEEE.

[188] SOBH, T., OWEN, J., JAYNES, C., DEKHIL, M., AND HENDERSON, T. Active inspection and reverse engineering. C.S. Dept. UUCS-93-007, University of Utah, Salt Lake City, Utah, USA, March 1993.

[189] SOBH, T., JAYNES, C., AND HENDERSON, T. A discrete event framework for intelligent inspection. In *Proc. 1993 IEEE Int. Conf. Robotics and Automation* (May 1993). Atlanta, GA.

[190] T. M. Sobh and K. Wohn, "Recovery of 3-D Motion and Structure by Temporal Fusion". In *Proceedings of the 2nd SPIE Conference on Sensor Fusion*, November 1989.

[191] X. Song and J. W. S. Liu, "Performance of multiversion concurrency control algorithms in maintaining temporal consistency," in *Proceedings of IEEE Compsac*, (Chicago, IL), Nov. 1990.

[192] J. A. Stankovic and K. Ramamritham, eds., *Tutorial: Hard Real-Time Systems.* Washington, D. C.: IEEE Computer Society Press, 1988.

[193] S. G. Strickland and C. G. Cassandras, "An "augmented chain" approach for on–line sensitivity analysis of Markov processes," in *Proceedings of 26th IEEE Conference on Decision and Control*, (Los Angeles, CA), pp. 1873–1878, Dec. 1987.

[194] M. Subbarao and A. M. Waxman, *On The Uniqueness of Image Flow Solutions for Planar Surfaces in Motion*, CAR–TR–113, Center for Automation Research, University of Maryland, April 1985.

[195] R. Suri, "Perturbation Analysis : The State of the Art and Research Issues Explained via the GI/G/1 Queue", *Proc. of the IEEE*, January 1989.

[196] A. Tomlinson, G. Hoagland, and V. K. Garg, "Distributed resource management using active supervisory predicate control," in *Proceedings of 1992 American Control Conference*, (Chicago, IL), June 1992.

[197] S. Ullman, "Analysis of Visual Motion by Biological and Computer Systems", *IEEE Computer*, August 1981.

[198] S. Ullman, *Maximizing Rigidity: The incremental recovery of 3-D structure from rigid and rubbery motion*, AI Memo 721, MIT AI lab. 1983.

[199] K. P. Valavanis and G. N. Saridis, *Intelligent Robotic Systems: Theory, Design and Applications.* Boston: Kluwer Academic Publishers, 1992.

[200] VAN THIEL, M. Feature based automated part inspection. Master's thesis, University of Utah, 1993.

[201] A. F. Vaz and W. M. Wonham, "On supervisor reduction in discrete event systems," *International Journal of Control*, vol. 44, no. 2, pp. 475–491, 1986.

[202] D. Vergamini, "Verification by means of observational equivalence on automata," Tech. Rep. 501, INRIA, Sophia-Antipolis, France, 1986.

[203] J. Walrand, *An Introduction to Queueing Networks.* Englewood Cliffs, NJ: Prentice–Hall, 1988.

[204] F. Y. Wang, "Supervisory control for concurrent discrete event dynamic systems based on petri nets," in *Proceedings of 31st IEEE Conference on Decision and Control*, pp. 1196–1197, Dec. 1992.

[205] Y. Wardi, W. B. Gong, C. G. Cassandras, and M. H. Kallmes, "Smoothed perturbation analysis for a class of piecewise constant sample performance functions," *Discrete Event Dynamic Systems: Theory and Applications*, vol. 1, no. 4, pp. 393–414, 1991.

[206] J. F. Watson and A. A. Desrochers, "Applying generalized stochastic Petri nets to manufacturing systems containing non-exponential transition functions," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 21, no. 5, pp. 1008–1017, 1991.

[207] N. Weiderman, "Hartstone: synthetic benchmark requirements for hard real–time applications," tech. rep., Software Engineering Institute, Carnegie Mellon Univ., Mar. 1989.

[208] D. S. Weld, *Theories of comparative analysis.* The MIT Press, 1990.

[209] G. Werling, "Planning of sensing tasks in an assembly environment," *Journal of Intelligent and Robotic Systems*, vol. 4, pp. 221–254, 1991.

[210] D. E. Whitney, "State space models of remote manipulation tasks," *IEEE Transactions on Automatic Control*, vol. 14, no. 6, pp. 617–623, 1969.

[211] W. Whitt, "Continuity of generalized semi–Markov processes," *Mathematics of Operations Research*, vol. 5, pp. 494–501, 1980.

[212] H. Wong–Toi and D. L. Dill, "Synthesizing processes and schedulers from temporal specifications," in *Computer-Aided Verification '90, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Volume 3*, pp. 177–186, American Mathematical Society, 1991.

[213] W. M. Wonham, *Computational and Combinatorial Methods in System Theory*, ch. On Control of Discrete Event Systems, pp. 159–174. North–Holland: Elsevier Science Publisher B.V., 1986.

[214] C. M. Woodside, "Response time sensitivity measurement for computer systems and general closed queueing networks," *J. Performance Evaluation*, vol. 4, pp. 199–210, 1984.

[215] Y. T. Yu and M. G. Gouda, "Deadlock detection for a class of communicating finite state machines," *IEEE Transactions on Communications*, vol. 30, no. 12, pp. 2512–2516, 1982.

[216] B. P. Zeigler, *Multifacetted modelling and discrete event simulation.* New York: Academic Press, 1984.

[217] B. P. Zeigler, "Hierarchical, modular discrete event modelling in an object oriented environment," *Simulation J.*, vol. 49, no. 5, pp. 219–230, 1987.

[218] B. Zhang, *Performance gradient estimation for very large Markov chains.* PhD thesis, Harvard Univ., 1990.

[219] W. Zhang, "Representation of assembly and automatic robot planning by Petri net," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 19, no. 2, pp. 418–422, 1989.

[220] H. Zhong and W. M. Wonham, "On the consistency of hierarchical supervision in discrete–event systems," *IEEE Transactions on Automatic Control*, vol. 35, no. 10, pp. 1125–1134, 1990.

[221] M. C. Zhou, *A Theory for the Synthesis and Augmentation of Petri Nets in Automation.* PhD thesis, ECSE, Rensselear Polytechnic Institute, Troy, NY, 1990.

[222] M. C. Zhou and F. DiCesare, *Petri Net Synthesis for Discrete Event Control of Manufacturing Systems.* Boston: Kluwer Academic Publishers, 1993.