Object-Oriented and Classical Software Engineering

CLASSICAL ANALYSIS

Overview

- The specification document
- Informal specifications
- Structured systems analysis
- Structured systems analysis: The MSG Foundation case study
- Other semiformal techniques
- Entity-relationship modeling
- Finite state machines
- Petri nets
- Z

Overview (contd)

- Other formal techniques
- Comparison of classical analysis techniques
- Testing during classical analysis
- CASE tools for classical analysis
- Metrics for classical analysis
- Software project management plan: The MSG Foundation case study
- Challenges of classical analysis

The Specification Document Must Be

- Informal enough for the client
  - The client is generally not a computer specialist
- Formal enough for the developers
  - It is the sole source of information for drawing up the design
- These two requirements are mutually contradictory

12.1 The Specification Document

- The specification document is a contract between the client and the developers

- Typical constraints
  - Deadline
  - Parallel running
  - Portability
  - Reliability
  - Rapid response time

- For real-time software
  - Hard real-time constraints must be satisfied

Specification Document (contd)

- Acceptance criteria
  - It is vital to spell out a series of tests
- If the product passes the tests, it is deemed have satisfied its specifications
- Some acceptance criteria are restatements of constraints

Solution Strategy

- A general approach to building the product
- Find strategies without worrying about constraints
  - Then modify the strategies in the light of the constraints, if necessary
- Keep a written record of all discarded strategies, and why they were discarded
  - To protect the analysis team
  - To prevent unwise new “solutions” during postdelivery maintenance

12.2 Informal Specifications

- Informal specifications are written in a natural language
  - Examples: English, Mandarin, Kiswahili, Hindi

- Example

  “If the sales for the current month are below the target sales, then a report is to be printed, unless the difference between target sales and actual sales is less than half of the difference between target sales and actual sales in the previous month, or if the difference between target sales and actual sales for the current month is under 5%”

The Meaning of This Specification

- The sales target for January was $100,000, but actual sales were only $64,000 (36% below target)
  - Print the report
- The sales target for February was $120,000, the actual sales were only $100,000 (16.7% below target)
  - The percentage difference for February (16.7%) is less than half of the previous month's percentage difference (36%), so do not print the report

The Meaning of This Specification (contd)

- The sales target for March was $100,000, the actual sales were $98,000 (2% below target)
  - The percentage difference is under 5%, so do not print the report

But the Specifications Do Not Say This

- "[D]ifference between target sales and actual sales"
  - There is no mention of percentage difference in the specifications
- The difference in January was $36,000, the difference in February was $20,000
  - Not less than half of $36,000, so the report is printed
- "[D]ifference … [of] 5%"
  - Again, no mention of percentage

But the Specifications Do Not Say This (contd)

- Ambiguity—should the last clause read "percentage difference … [of] 5%" or "difference … [of] $5,000" or something else entirely?

- The style is poor
  - The specifications should state when the report should be printed …
  - … Rather than when it should not be printed

Informal Specifications (contd)

- Claim
  - This cannot arise with professional specifications writers
- Refutation
  - Text processing case study

12.2.1 Correctness Proof Case Study

- Naur text-processing problem

  Given a text consisting of words separated by `blank` or by `newline` characters, convert it to line-by-line form in accordance with the following rules:

  (1) line breaks must be made only where the given text contains a `blank` or `newline`;

  (2) each line is filled as far as possible, as long as

  (3) no line will contain more than `maxpos` characters

Episode 1

- 1969 — Naur Paper
- Naur constructed a procedure (25 lines of Algol 60), and informally proved its correctness

Episode 2

- 1970 — Reviewer in *Computing Reviews*
  - The first word of the first line is preceded by a `blank` unless the first word is exactly `maxpos` characters long

Episode 3

- 1971 — London found 3 more faults
  - Including: The procedure does not terminate unless a word longer than `maxpos` characters is encountered

Episode 4

- 1975 — Goodenough and Gerhart found 3 further faults
  - Including: The last word will not be output unless it is followed by a `blank` or `newline`
- Goodenough and Gerhart then produced a new set of specifications, about four times longer than Naur's

Episode 5

- 1985 — Meyer detected 12 faults in Goodenough and Gerhart’s specifications

Episode 5

- Goodenough and Gerhart's specifications
  - Were constructed with the greatest of care
  - Were constructed to correct Naur's specifications
  - Went through two versions, carefully refereed
  - Were written by experts in specifications,
  - With as much time as they needed,
  - For a product about 30 lines long
- So, what chance do we have of writing fault-free specifications for a real product?

Episode 6

- 1989 — Schach found a fault in Meyer's specifications
  - Item (2) of Naur's original requirement ("each line is filled as far as possible") is not satisfied

Informal Specifications (contd)

- Conclusion
  - Natural language is *not* a good way to specify a product

12.3 Structured Systems Analysis

- Three popular graphical specification methods of 1970s
  - DeMarco
  - Gane and Sarsen
  - Yourdon
- All are equivalent
- All are equally good

12.3 Structured Systems Analysis (contd)

- Many U.S. corporations use them for commercial products
- Gane and Sarsen’s method is taught here because it is so widely used

12.3.1 Sally's Software Shop Mini Case Study

- Sally's Software Shop buys software from various suppliers and sells it to the public. Popular software packages are kept in stock, but the rest must be ordered as required. Institutions and corporations are given credit facilities, as are some members of the public. Sally's Software Shop is doing well, with a monthly turnover of 300 packages at an average retail cost of $250 each. Despite her business success, Sally has been advised to computerize. Should she?

Sally's Software Shop Mini Case Study (contd)

- A better question
  - What business functions should she computerize
    - » Accounts payable
    - » Accounts receivable
    - » Inventory
- Still better
  - How? Batch, or online? In-house or outsourcing?

Sally's Software Shop Mini Case Study (contd)

- The fundamental issue
  - What is Sally's objective in computerizing her business?
- Because she sells software?
  - She needs an in-house system with sound and light effects
- Because she uses her business to launder "hot" money?
  - She needs a product that keeps five different sets of books, and has no audit trail

Sally's Software Shop Mini Case Study (contd)

- We assume: Sally wishes to computerize "in order to make more money"
  - We need to perform cost–benefit analysis for each section of her business

Sally's Software Shop Mini Case Study (contd)

- The danger of many standard approaches
  - First produce the solution, then find out what the problem is!
- Gane and Sarsen's method
  - Nine-step method
  - Stepwise refinement is used in many steps

Sally's Software Shop Mini Case Study (contd)

- The data flow diagram (DFD) shows the logical data flow
  - "What happens, not how it happens"
- Symbols:

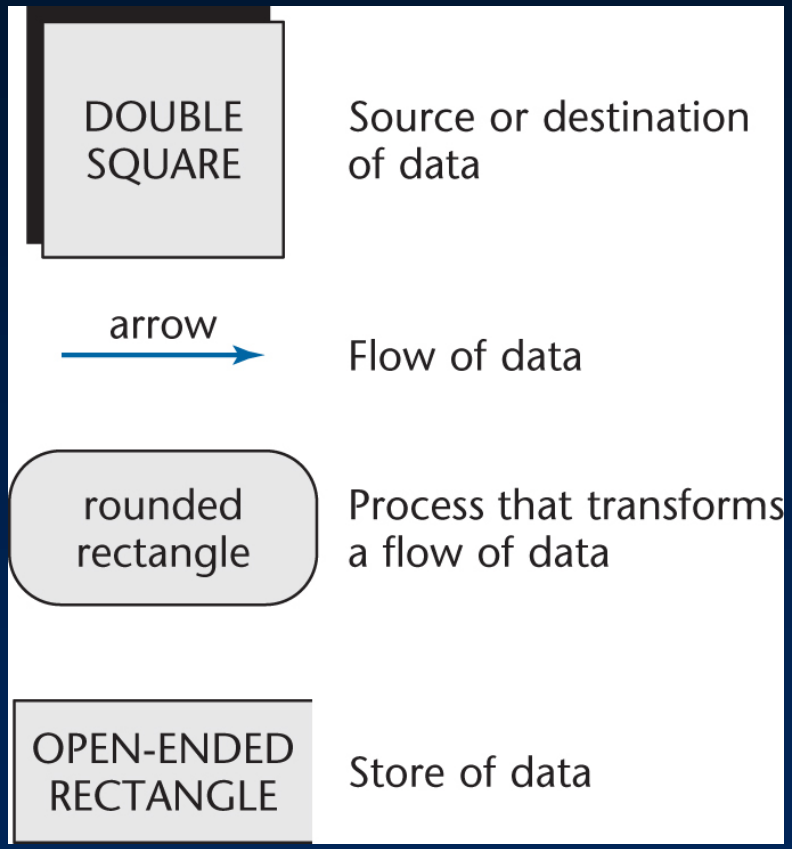

Figure 12.1

Step 1. Draw the DFD

- First refinement
  - Infinite number of possible interpretations

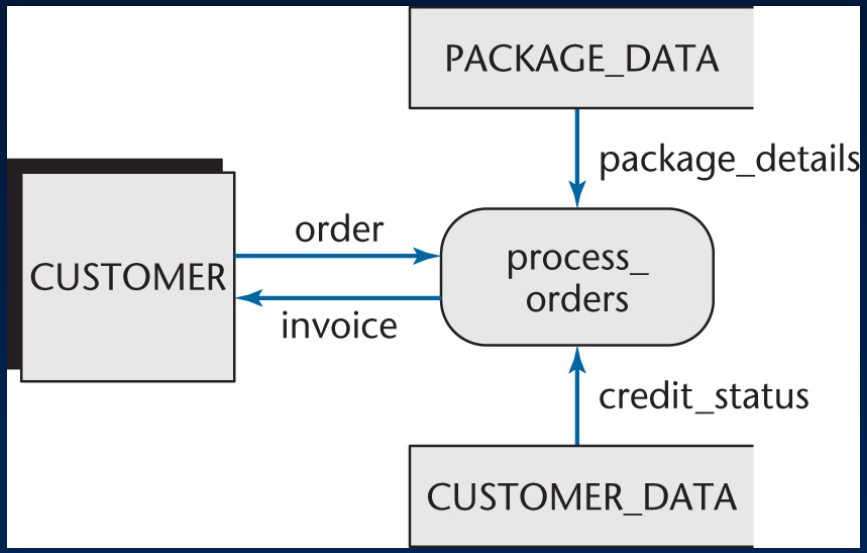

Figure 12.2

Step 1 (contd)

- Second refinement
  - PENDING ORDERS is scanned daily

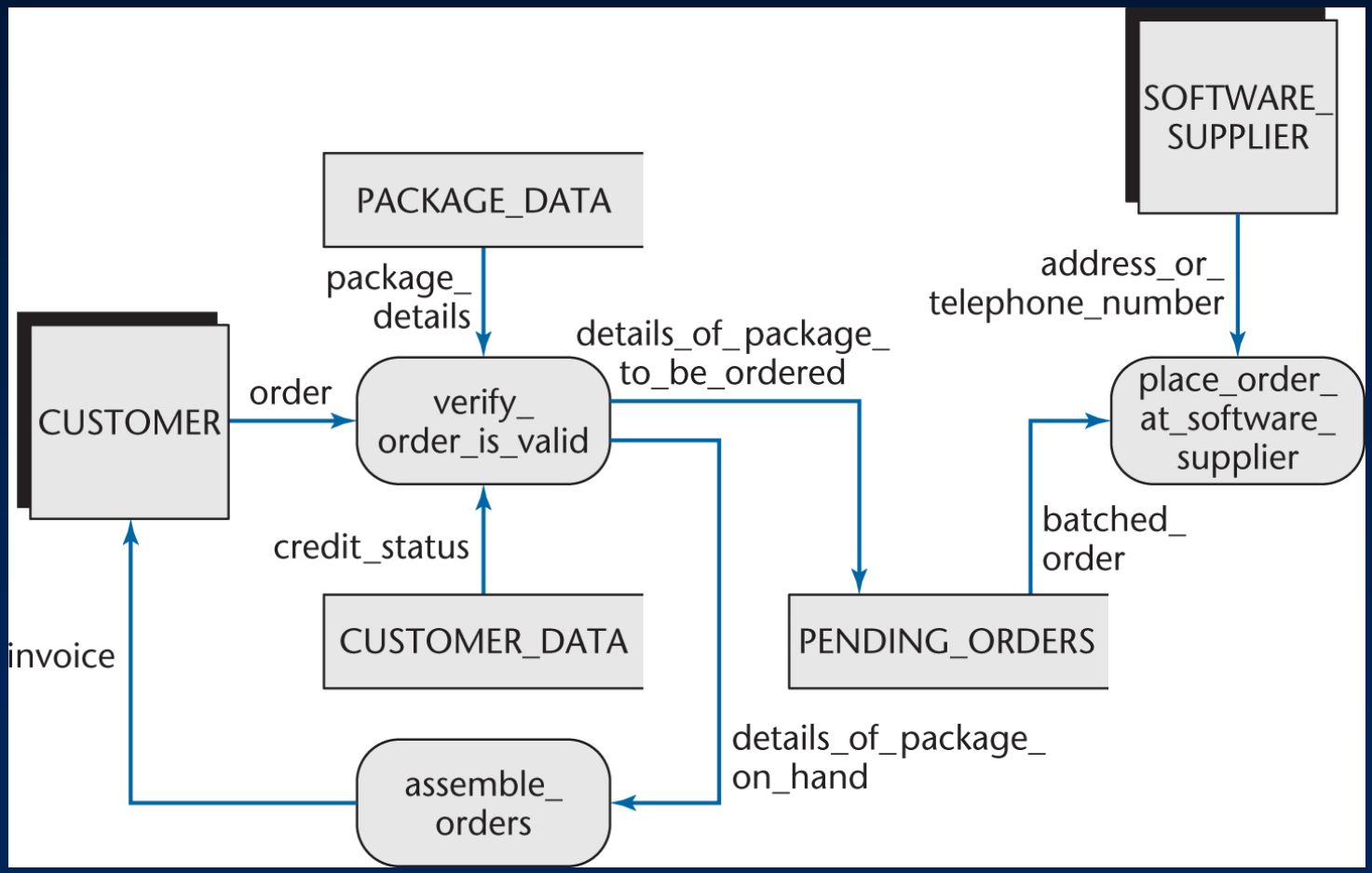

Figure 12.3

Step 1 (contd)

- Portion of third refinement

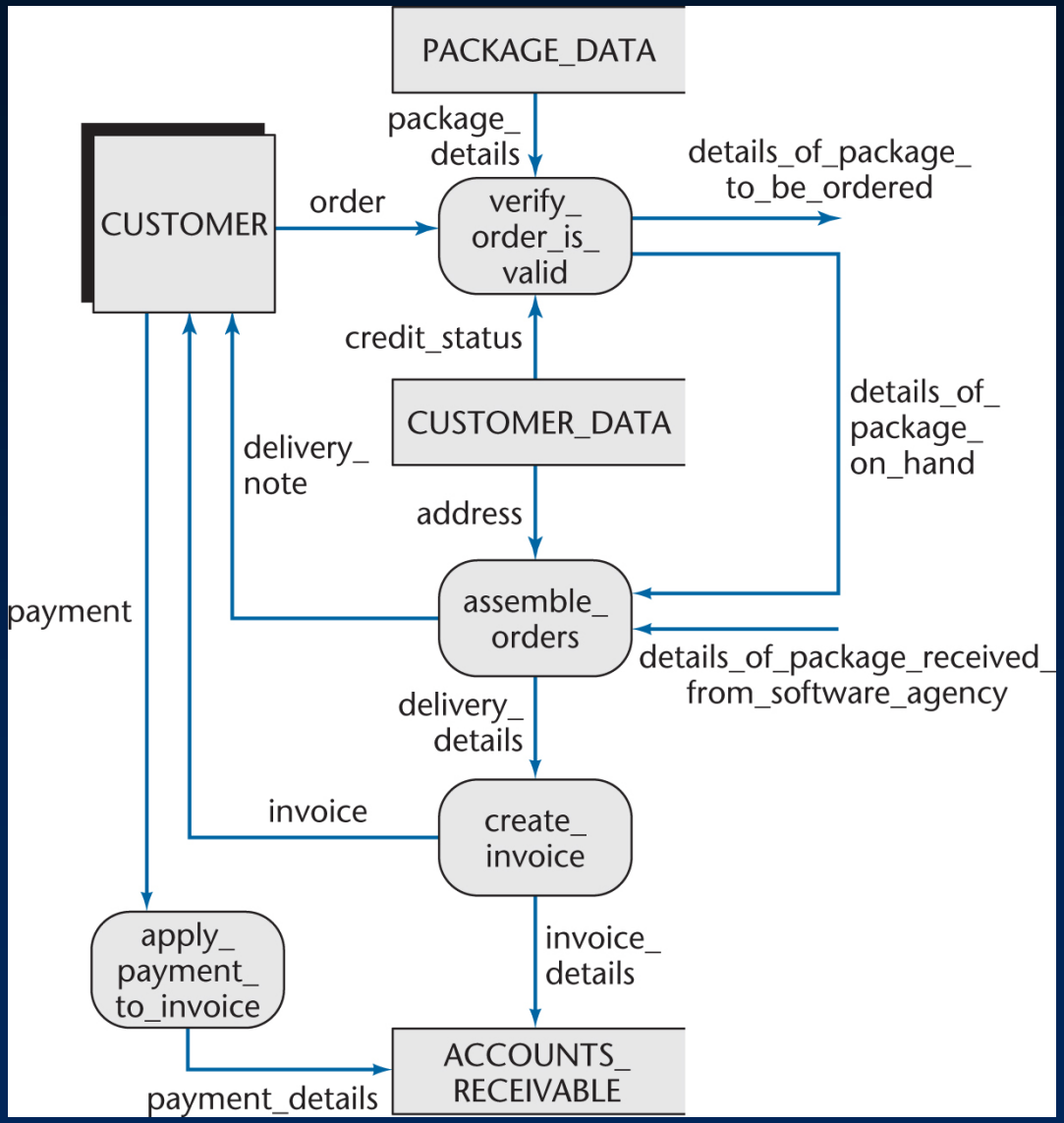

Figure 12.4

Step 1 (contd)

- The final DFD is larger
  - But it is easily understood by the client
- When dealing with larger DFDs
  - Set up a hierarchy of DFDs
  - A box becomes a DFD at a lower level

Step 2. Decide What Parts to Computerize and How

- It depends on how much client is prepared to spend
- Large volumes, tight controls
  - Batch
- Small volumes, in-house microcomputer
  - Online
- Cost/benefit analysis

Step 3. Determine the Details of the Data Flows

- Determine the data items for each data flow
- Refine each flow stepwise
  - Example;

```
order:
   order_identification
   customer_details
   package_details
```

- We need a data dictionary for larger products

Sample Data Dictionary Entries

Name of Data Element	Description	Narrative
order	Record comprising fields order_identification customer_details customer_name customer_address . . . package_details package_name package_price . . .	The fields contain all details of an order
order_identification	12-digit integer	Unique number generated by procedure generate_order_number. The first 10 digits contain the order number itself, the last 2 digits are check digits.
verify_order_is_valid	Procedure: Input parameter: order Output parameter: number_of_errors	This procedure takes order as input and checks the validity of every field; for each error found, an appropriate message is displayed on the screen (the total number of errors found is returned in parameter number_of_errors).

Figure 12.5

Step 4. Define the Logic of the Processes

- We have process `give educational discount`
  - Sally must explain the discount she gives to educational institutions
    - » 10% on up to 4 packages
    - » 15% on 5 or more

Step 4 . Define the Logic of the Processes (contd)

- Translate this into a decision tree

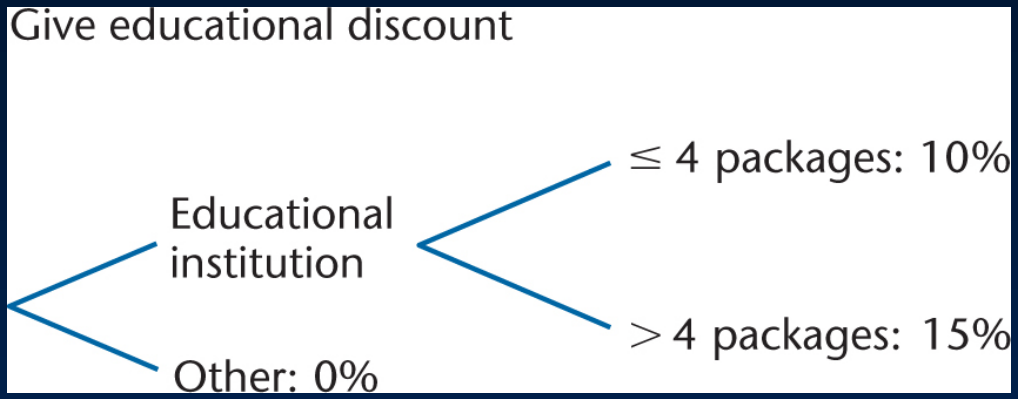

Figure 12.6

Step 4. Define the Logic of the Processes (contd)

- The advantage of a decision tree
  - Missing items are quickly apparent

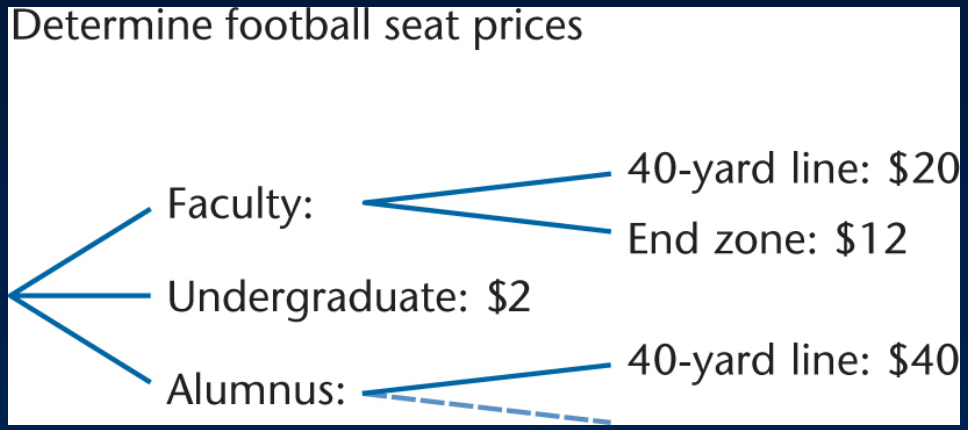

Figure 12.7

Step 5. Define the Data Stores

- Define the exact contents and representation (format)
  - COBOL: specify to `pic` level
  - Ada: specify `digits` or `delta`

Step 5. Define the Data Stores (contd)

- Specify where immediate access is required
  - Data immediate-access diagram (DIAD)

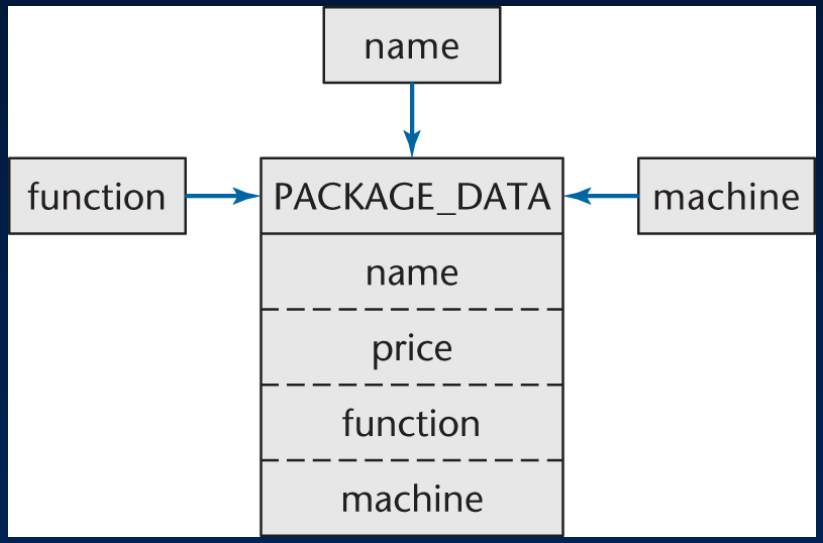

Figure 12.8

Step 6. Define the Physical Resources

- For each file, specify
  - File name
  - Organization (sequential, indexed, etc.)
  - Storage medium
  - Blocking factor
  - Records (to field level)
  - Table information, if a DBMS is to be used

Step 7. Determine Input/Output Specifications

- Specify
  - Input forms
  - Input screens
  - Printed output

Step 8. Determine the Sizing

- Obtain the numerical data needed in Step 9 to determine the hardware requirements
  - Volume of input (daily or hourly)
  - Size, frequency, deadline of each printed report
  - Size, number of records passing between CPU and mass storage
  - Size of each file

Step 9. Determine the Hardware Requirements

- Mass storage requirements
- Mass storage for back-up
- Input needs
- Output devices
- Is the existing hardware adequate?
  - If not, recommend whether to buy or lease additional hardware

However

- Response times cannot be determined
- The number of I/O channels can only be guessed
- CPU size and timing can only be guessed
- Nevertheless, no other method provides these data for arbitrary products

Overall

- The method of Gane and Sarsen/De Marco/ Yourdon has resulted in major improvements in the software industry

12.4 Structured Systems Analysis: The MSG Foundation Case Study

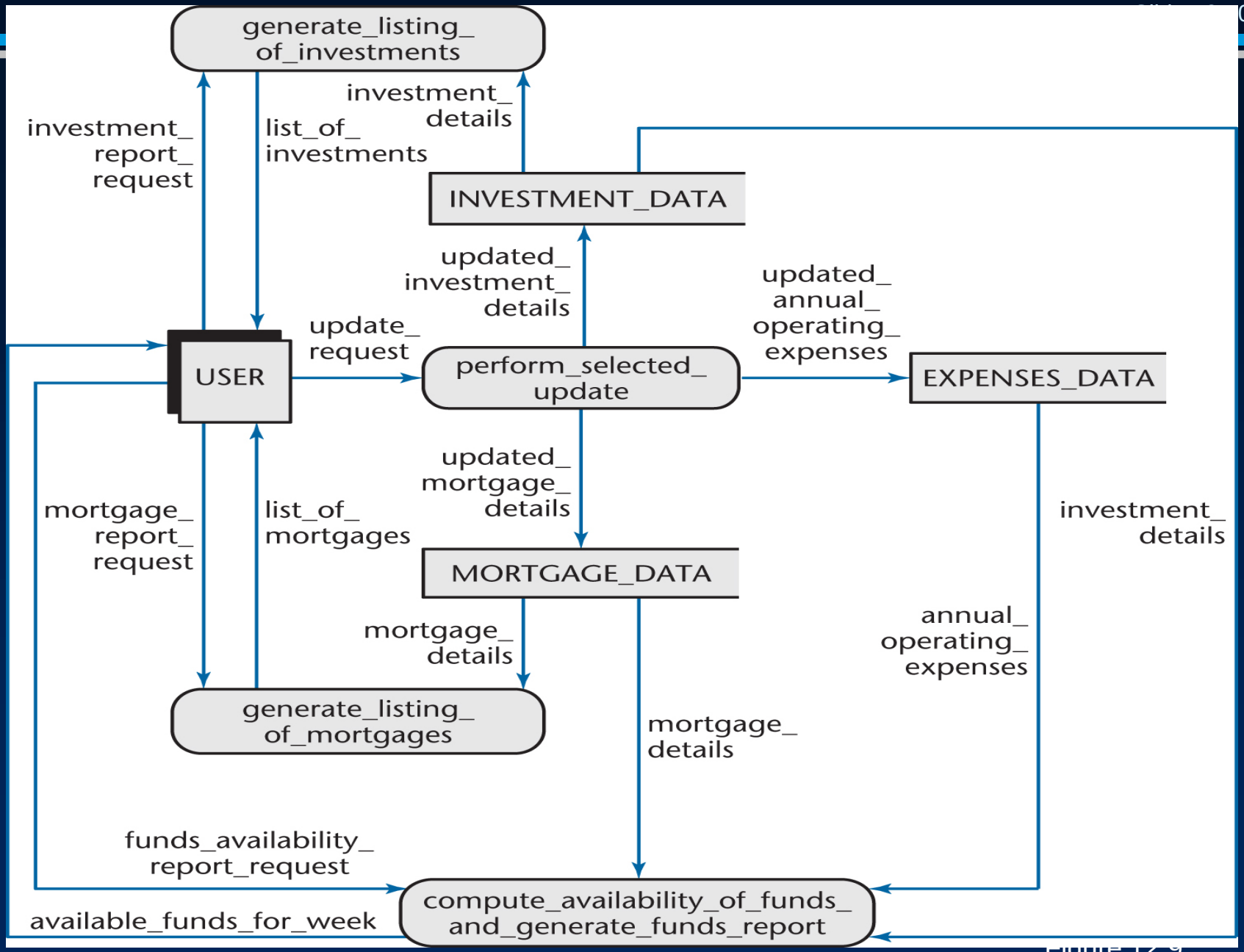

Figure 12.9

- As reflected in the DFD, the user can perform three different types of operations:

- 1. Update investment data, mortgage data, or operating expenses data:
  - The `USER` enters an `update_request`
  - To update investment data, process `perform_selected_update` solicits the `updated_investment_details` from the `USER`, and sends then to the `INVESTMENT_DATA` store of data
  - Updating mortgage data or expenses data is similar

- 2. Print a listing of investments or mortgages:
  - To print a list of investments, the `USER` enters an `investment_report_request`. Process `generate_listing_of_investments` then obtains investment data from store `INVESTMENT_DATA`, formats the report, and then prints the report
  - Printing a listing of mortgages is similar

- 3. Print a report showing available funds for mortgages for the week:
  - The `USER` enters a `funds_availability_report_request`.
  - To determine how much money is available for mortgages for the current week, process `compute_availability_of_funds_and_generate_funds_report` then obtains (see next slide):

- `investment_details` from store `INVESTMENT_DATA` and computes the expected total annual return on investments
- `mortgage_details` from store `MORTGAGE_DATA` and computes the expected income for the week, expected mortgage payments for the week, and expected grants for the week
- `annual_operating_expenses` from store `EXPENSES_DATA` and computes the expected annual operating expense

- Process `compute_availability_of_funds_and_generate_funds_report` then uses these results to compute `available_funds_for_week`, and format and print the report

12.5 Other Semiformal Techniques

- Semiformal (graphical) techniques for classical analysis include
  - PSL/PSA
  - SADT
  - SREM

12.6 Entity-Relationship Modeling

- Semi-formal technique
  - Widely used for specifying databases
  - Example:

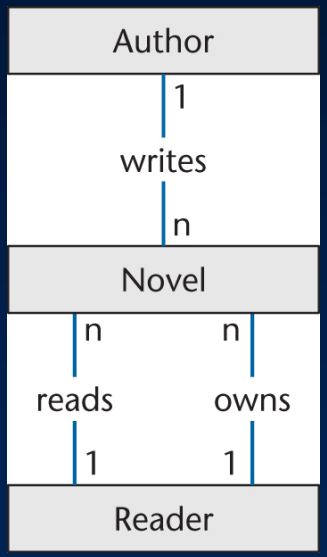

Figure 12.10

Entity-Relationship Diagrams (contd)

- Many-to-many relationship

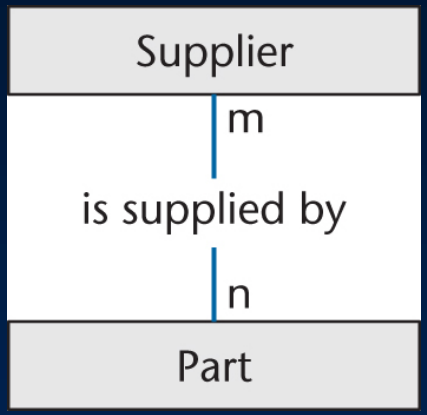

Figure 12.11

Entity-Relationship Diagrams (contd)

- More complex entity-relationship diagrams

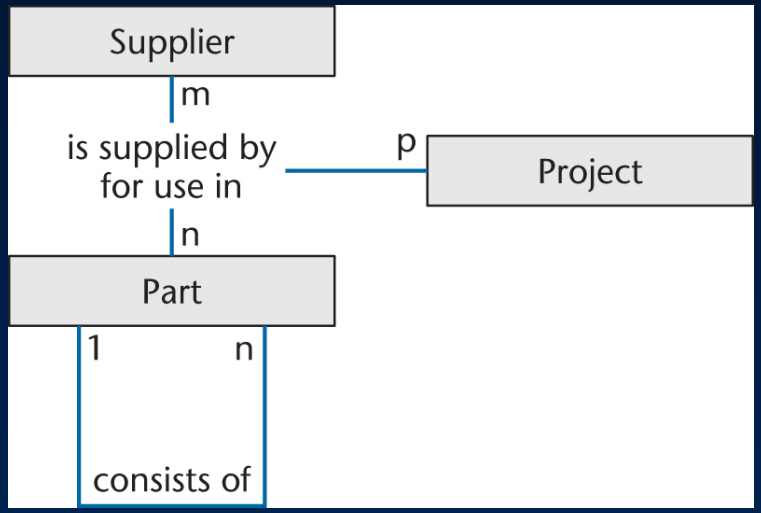

Figure 12.12

12.7 Finite State Machines

- Case study

  A safe has a combination lock that can be in one of three positions, labeled 1, 2, and 3. The dial can be turned left or right (L or R). Thus there are six possible dial movements, namely 1L, 1R, 2L, 2R, 3L, and 3R. The combination to the safe is 1L, 3R, 2L; any other dial movement will cause the alarm to go off

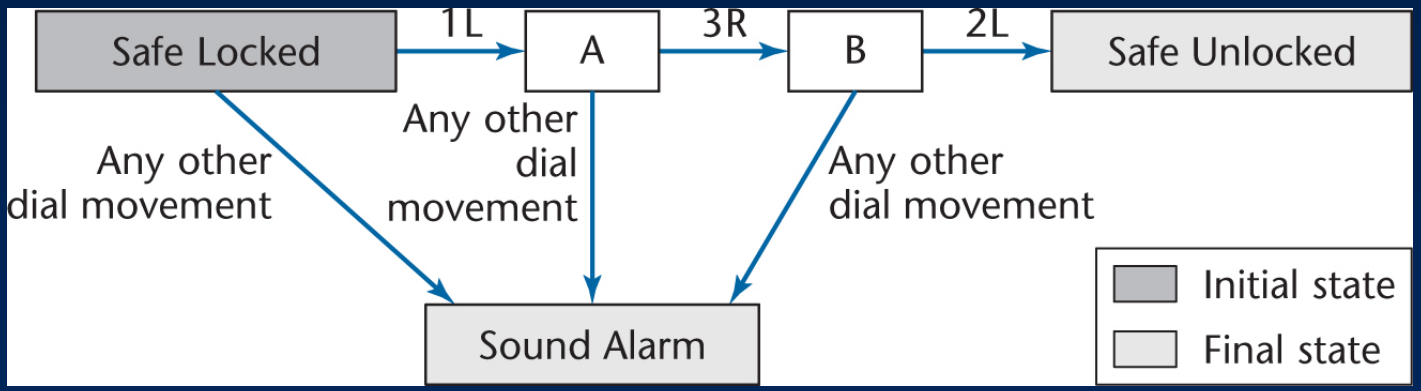

Figure 12.13

Finite State Machines (contd)

- The set of states J is {Safe Locked, A, B, Safe Unlocked, Sound Alarm}
- The set of inputs K is {1L, 1R, 2L, 2R, 3L, 3R}
- The transition function T is on the next slide
- The initial state J is Safe Locked
- The set of final states J is {Safe Unlocked, Sound Alarm}

Finite State Machines (contd)

- Transition table

Dial Movement \ Current State	Table of Next States Safe Locked	A	B
1L	A	Sound alarm	Sound alarm
1R	Sound alarm	Sound alarm	Sound alarm
2L	Sound alarm	Sound alarm	Safe unlocked
2R	Sound alarm	Sound alarm	Sound alarm
3L	Sound alarm	Sound alarm	Sound alarm
3R	Sound alarm	B	Sound alarm

Figure 12.14

Extended Finite State Machines

- FSM transition rules have the form

  current state [menu] and **event** [option selected] $\Rightarrow$ **new state**

- Extend the standard FSM by adding global predicates

- Transition rules then take the form

  current state and **event** and **predicate** $\Rightarrow$ **new state**

12.7.1 Finite State Machines: The Elevator Problem Case Study

A product is to be installed to control n elevators in a building with m floors. The problem concerns the logic required to move elevators between floors according to the following constraints:

1. Each elevator has a set of m buttons, one for each floor. These illuminate when pressed and cause the elevator to visit the corresponding floor. The illumination is canceled when the corresponding floor is visited by the elevator
2. Each floor, except the first and the top floor, has two buttons, one to request an up-elevator, one to request a down-elevator. These buttons illuminate when pressed. The illumination is canceled when an elevator visits the floor, then moves in the desired direction
3. If an elevator has no requests, it remains at its current floor with its doors closed

Finite State Machines: The Elevator Problem Case Study (contd)

- There are two sets of buttons
- Elevator buttons
  - In each elevator, one for each floor
- Floor buttons
  - Two on each floor, one for up-elevator, one for down-elevator

Elevator Buttons

- EB (e, f):
  - Elevator Button in elevator e pressed to request floor f

Elevator Buttons (contd)

- Two states

  EBON (e, f): Elevator Button (e, f) ON

  EBOFF (e, f): Elevator Button (e, f) OFF

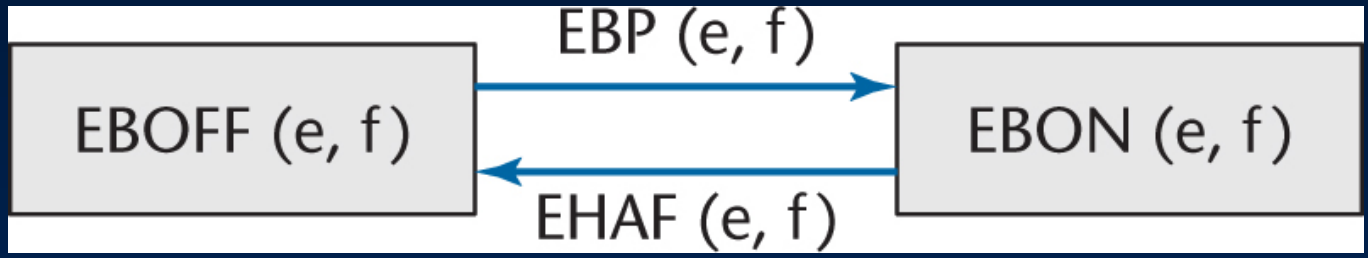

Figure 12.15

  - If an elevator button is on and the elevator arrives at floor f, then the elevator button is turned off
  - If the elevator button is off and the elevator button is pressed, then the elevator button comes on

Elevator Buttons (contd)

- Two events

  EBP (e, f): Elevator Button (e, f) Pressed

  EHAF (e, f): Elevator e Has Arrived at Floor f

- Global predicate

  V (e, f): Elevator e is Visiting (stopped at) floor f

- Transition Rules

  EBOFF (e, f) and EBP (e, f) and not V (e, f) $\Rightarrow$ EBON (e, f)

  EBON (e, f) and EHAF (e, f) $\Rightarrow$ EBOFF (e, f)

Floor Buttons

- FB (d, f):
  - Floor Button on floor f that requests elevator traveling in direction d

Floor Buttons (contd)

- States

  FBON (d, f): Floor Button (d, f) ON

  FBOFF (d, f): Floor Button (d, f) OFF

Figure 12.16

  - If the floor button is on and an elevator arrives at floor f, traveling in the correct direction d, then the floor button is turned off
  - If the floor button is off and the floor button is pressed, then the floor button comes on

Floor Buttons (contd)

- Events

  FBP (d, f): Floor Button (d, f) Pressed

  EHAF (1..n, f): Elevator 1 or … or n Has Arrived at Floor f

- Predicate

  S (d, e, f): Elevator e is visiting floor f

  Direction of motion is up (d = U), down (d = D), or no requests are pending (d = N)

- Transition rules

  FBOFF (d, f) and FBP (d, f) and not S (d, 1..n, f) $\Rightarrow$ FBON (d, f)

  FBON (d, f) and EHAF (1..n, f) and S (d, 1..n, f) $\Rightarrow$ FBOFF (d, f), d = U or D

Finite State Machines: The Elevator Problem Case Study (contd)

- The state of the elevator consists of component substates, including:
  - Elevator slowing
  - Elevator stopping
  - Door opening
  - Door open with timer running
  - Door closing after a timeout

Elevator Problem: FSM (contd)

- We assume that the elevator controller moves the elevator through the substates

- Three elevator states

  M (d, e, f): Moving in direction d (floor f is next)

  S (d, e, f): Stopped (d-bound) at floor f

  W (e, f): Waiting at floor f (door closed)

- For simplicity, the three stopped states S (U, e, f), S (N, e, f), and S (D, e, f) are grouped into one larger state

State Transition Diagram for Elevator

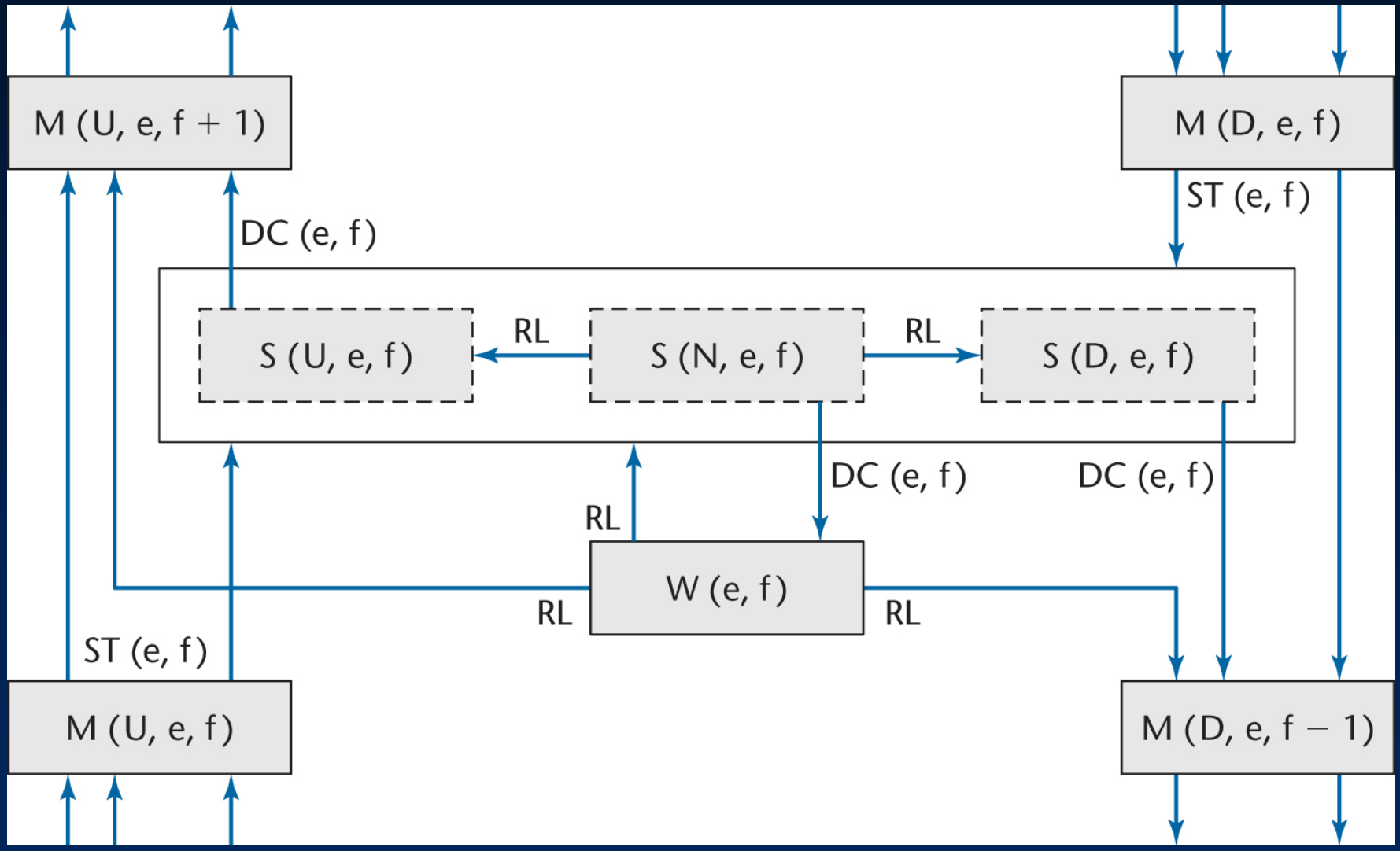

Figure 12.17

Elevator Problem: FSM (contd)

- Events

  DC (e, f): Door Closed for elevator e, floor f

  ST (e, f): Sensor Triggered as elevator e nears floor f

  RL: Request Logged (button pressed)

- Transition Rules

  If the elevator e is in state S (d, e, f) (stopped, d-bound, at floor f), and the doors close, then elevator e will move up, down, or go into the wait state

  DC (e, f) and S (U, e, f) $\Rightarrow$ M (U, e, f+1)

  DC (e, f) and S (D, e, f) $\Rightarrow$ M (D, e, f-1)

  DC (e, f) and S (N, e, f) $\Rightarrow$ W (e, f)

Finite State Machines (contd)

- The power of an FSM to specify complex systems
- There is no need for complex preconditions and postconditions
- Specifications take the simple form

  current state and **event** and **predicate** $\Rightarrow$ **next state**

Finite State Machines (contd)

- Using an FSM, a specification is
  - Easy to write down
  - Easy to validate
  - Easy to convert into a design
  - Easy to convert into code automatically
  - More precise than graphical methods
  - Almost as easy to understand
  - Easy to maintain
- However
  - Timing considerations are not handled

Finite State Machines (contd)

- Statecharts are a real-time extension of FSMs
  - CASE tool: Rhapsody

12.8 Petri Nets

- A major difficulty with specifying real-time systems is timing
  - Synchronization problems
  - Race conditions
  - Deadlock

- Often a consequence of poor specifications

Petri Nets (contd)

- Petri nets
  - A powerful technique for specifying systems that have potential problems with interrelations

- A Petri net consists of four parts:
  - A set of places P
  - A set of transitions T
  - An input function I
  - An output function O

Petri Nets (contd)

- Set of places P is $\{p_1, p_2, p_3, p_4\}$
- Set of transitions T is $\{t_1, t_2\}$
- Input functions:
  $I(t_1) = \{p_2, p_4\}$
  $I(t_2) = \{p_2\}$
- Output functions:
  $O(t_1) = \{p_1\}$
  $O(t_2) = \{p_3, p_3\}$

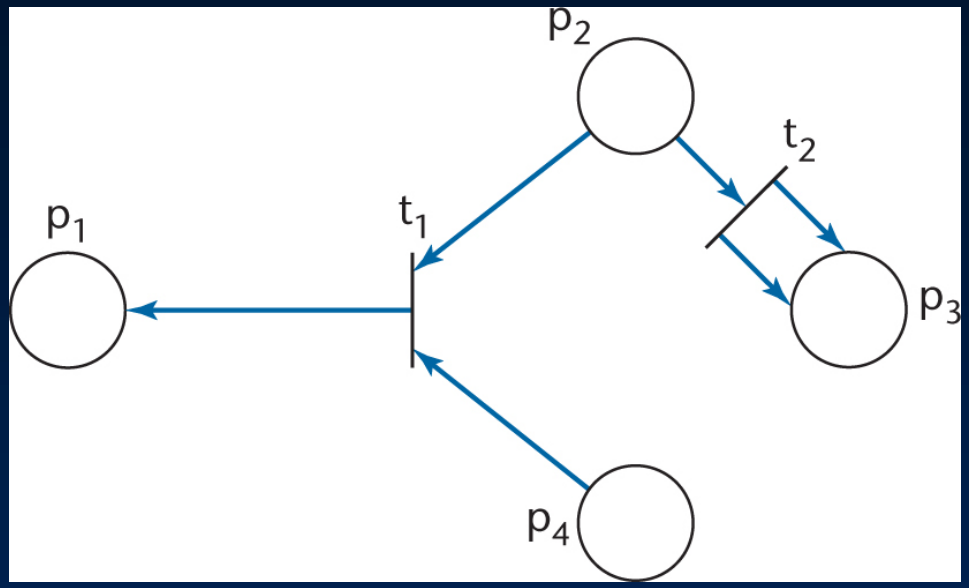

Figure 12.18

Petri Nets (contd)

- More formally, a Petri net is a 4-tuple $C = (P, T, I, O)$
  - $P = \{p_1, p_2, \ldots, p_n\}$ is a finite set of *places*, $n \geq 0$
  - $T = \{t_1, t_2, \ldots, t_m\}$ is a finite set of *transitions*, $m \geq 0$, with P and T disjoint
  - $I : T \rightarrow P^{\infty}$ is the *input* function, a mapping from transitions to bags of places
  - $O : T \rightarrow P^{\infty}$ is the *output* function, a mapping from transitions to bags of places
  - (A *bag* is a generalization of a set that allows for multiple instances of elements, as in the example on the previous slide)
  - A *marking* of a Petri net is an assignment of tokens to that Petri net

Petri Nets (contd)

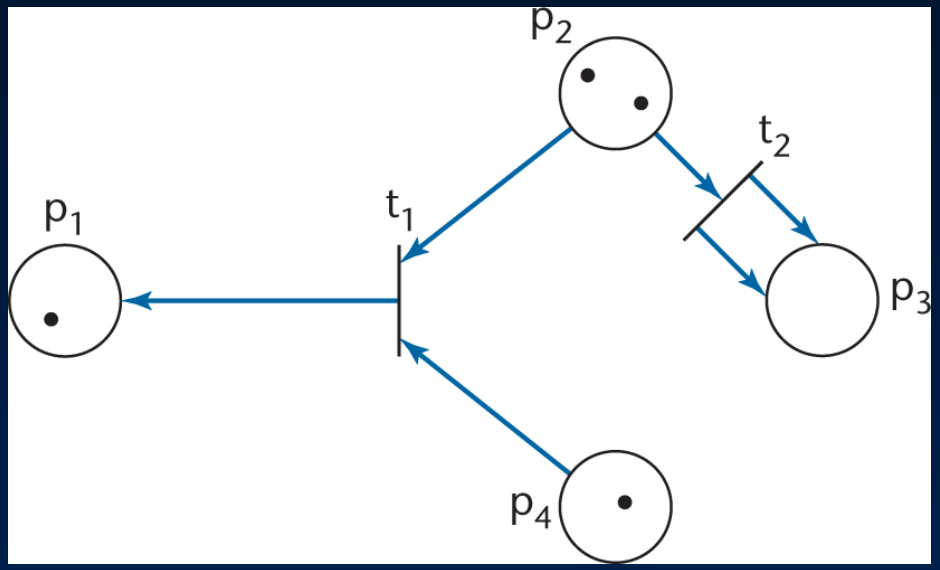

Figure 12.19

- Four tokens: one in p_1, two in p_2, none in p_3, and one in p_4
  - Represented by the vector (1, 2, 0, 1)

Petri Nets (contd)

- A transition is enabled if each of its input places has as many tokens in it as there are arcs from the place to that transition

Petri Nets (contd)

- Transition t_1 is enabled (ready to fire)
  - If t_1 fires, one token is removed from p_2 and one from p_4, and one new token is placed in p_1
- Transition t_2 is also enabled
- Important:
  - The number of tokens is not conserved

Petri Nets (contd)

- Petri nets are indeterminate
  - Suppose t_1 fires

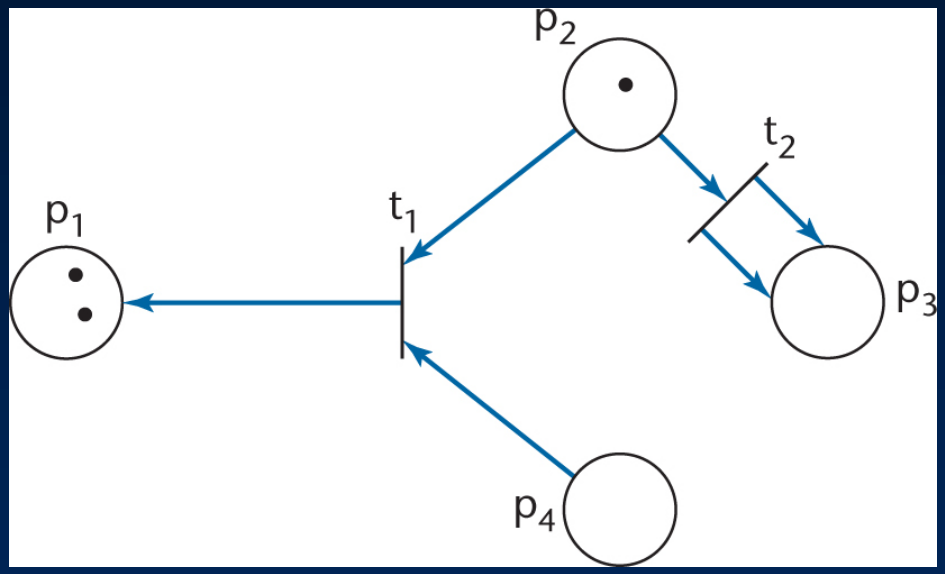

Figure 12.20

- The resulting marking is (2, 1, 0 ,0)

Petri Nets (contd)

- Now only t_2 is enabled
  - It fires

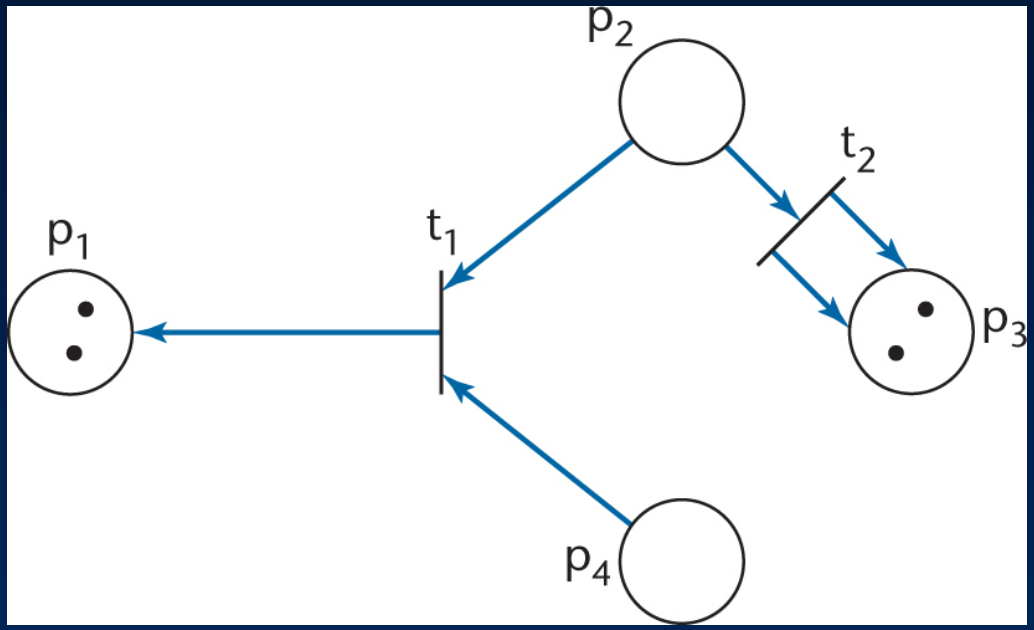

Figure 12.21

- The marking is now (2, 0, 2, 0)

Petri Nets (contd)

- More formally, a marking M of a Petri net

  $C = (P, T, I, O)$

  is a function from the set of places P to the non-negative integers

  $M : P \rightarrow \{0, 1, 2, \ldots\}$

- A marked Petri net is then a 5-tuple (P, T, I, O, M)

Petri Nets (contd)

- Inhibitor arcs
  - An inhibitor arc is marked by a small circle, not an arrowhead

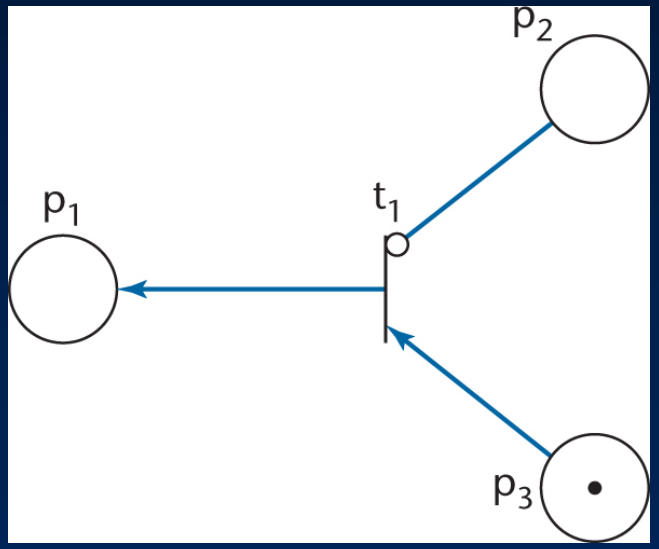

Figure 12.22

- Transition t_1 is enabled

Petri Nets (contd)

- In general, a transition is enabled if there is at least one token on each (normal) input arc, and no tokens on any inhibitor input arcs

12.8.1 Petri Nets: The Elevator Problem Case Study

- A product is to be installed to control n elevators in a building with m floors
- Each floor is represented by a place F_f, $1 \leq f \leq m$
- An elevator is represented by a token
- A token in F_f denotes that an elevator is at floor F_f

Petri Nets: The Elevator Problem Case Study (contd)

- First constraint:

  1. Each elevator has a set of m buttons, one for each floor. These illuminate when pressed and cause the elevator to visit the corresponding floor. The illumination is canceled when the corresponding floor is visited by an elevator

- The elevator button for floor f is represented by place EB_f, $1 \leq f \leq m$

- A token in EB_f denotes that the elevator button for floor f is illuminated

Petri Nets: The Elevator Problem Case Study (contd)

- A button must be illuminated the first time the button is pressed and subsequent button presses must be ignored

Petri Nets: The Elevator Problem Case Study (contd)

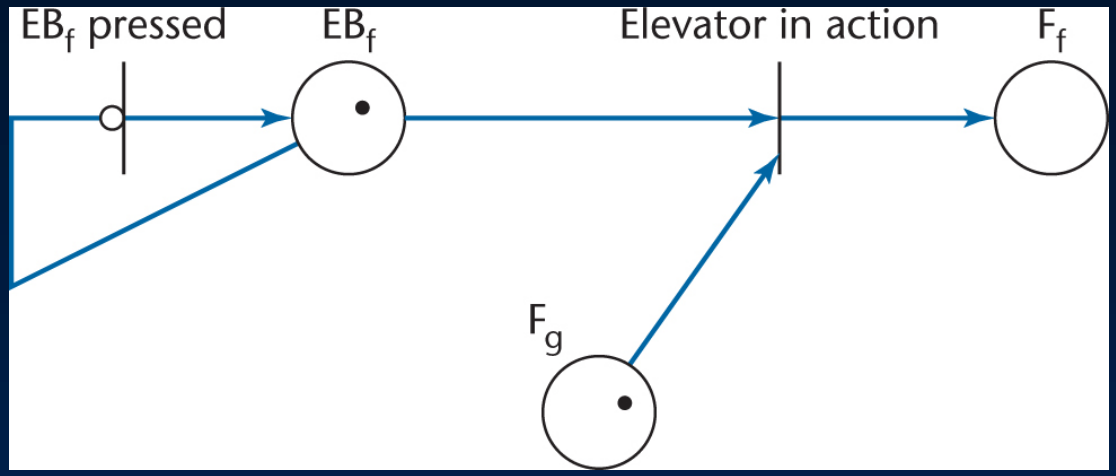

Figure 12.23

- If button EB_f is not illuminated, no token is in place and transition EB_f pressed is enabled
  - The transition fires, a new token is placed in EB_f
- Now, no matter how many times the button is pressed, transition EB_f pressed cannot be enabled

Petri Nets: The Elevator Problem Case Study (contd)

- When the elevator reaches floor g
  - A token is in place F_g
  - Transition Elevator in action is enabled, and then fires
- The tokens in EB_f and F_g are removed
  - This turns off the light in button EB_f
- A new token appears in F_f
  - This brings the elevator from floor g to floor f

Petri Nets: The Elevator Problem Case Study (contd)

- Motion from floor g to floor f cannot take place instantaneously
  - We need timed Petri nets

Petri Nets: The Elevator Problem Case Study (contd)

- Second constraint:

  2. Each floor, except the first and the top floor, has two buttons, one to request an up-elevator, one to request a down-elevator. These buttons illuminate when pressed. The illumination is canceled when the elevator visits the floor, and then moves in desired direction

- Floor buttons are represented by places FB^u_f and FB^d_f

Petri Nets: The Elevator Problem Case Study (contd)

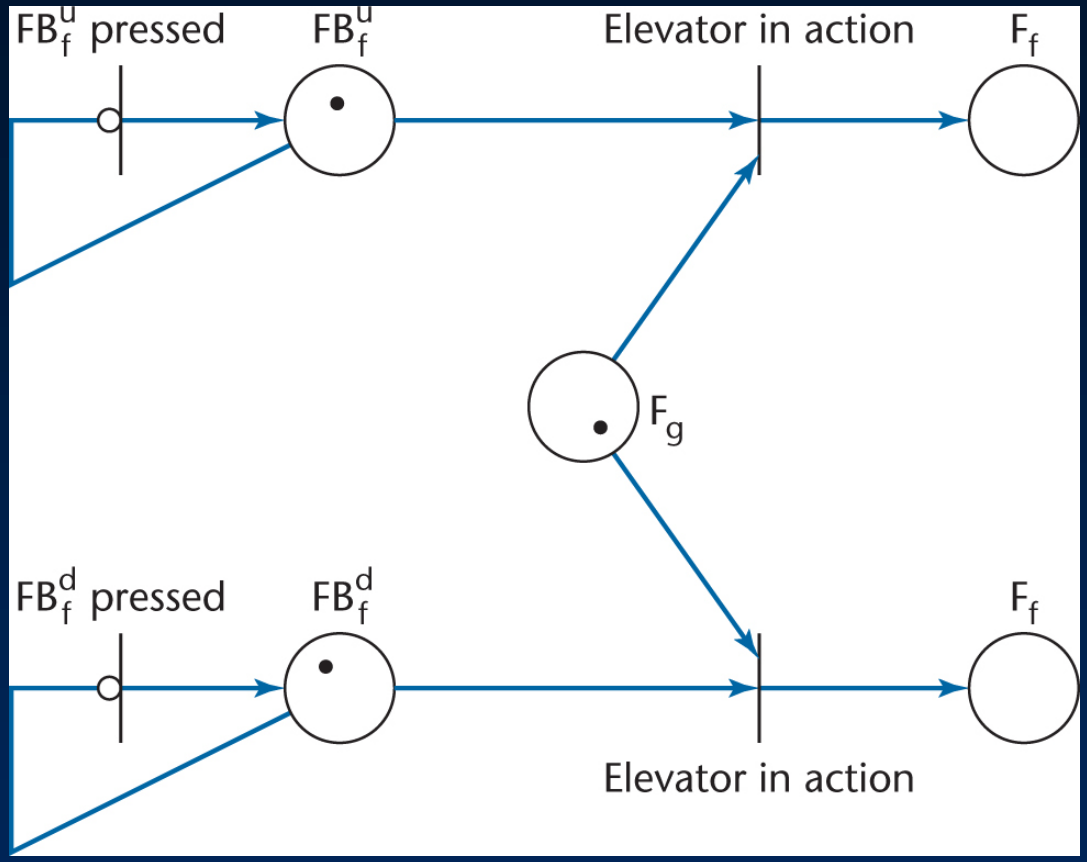

Figure 12.24

Petri Nets: The Elevator Problem Case Study (contd)

- The Petri net in the previous slide models the situation when an elevator reaches floor f from floor g with one or both buttons illuminated
- If both buttons are illuminated, only one is turned off
- A more complex model is needed to ensure that the correct light is turned off

Petri Nets: The Elevator Problem Case Study (contd)

- Third constraint:

  C_3. If an elevator has no requests, it remains at its current floor with its doors closed

- If there are no requests, no Elevator in action transition is enabled

Petri Nets: The Elevator Problem Case Study (contd)

- Petri nets can also be used for design
- Petri nets possess the expressive power necessary for specifying synchronization aspects of real-time systems

12.9 Z

- Z (pronounced “zed”) is a formal specification language
- There is a high squiggle factor

12.9.1 Z: The Elevator Problem Case Study

- A Z specification consists of four sections:
  - 1. Given sets, data types, and constants
  - 2. State definition
  - 3. Initial state
  - 4. Operations

1. Given sets

- Given sets are sets that need not be defined in detail
- Names appear in brackets
- Here we need the set of *all* buttons
- The specification therefore begins

  [Button]

2. State Definition

- Z specification consists of a number of schemata
  - A schema consists of a group of variable declarations, plus
  - A list of predicates that constrain the values of variables

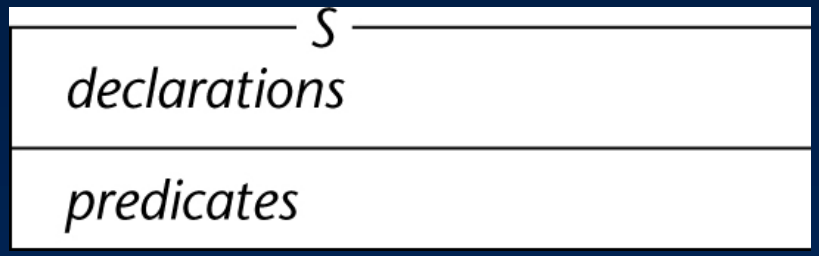

Figure 12.25

Z: The Elevator Problem Case Study (contd)

- In this problem there are four subsets of Button
  - The floor buttons
  - The elevator buttons
  - buttons (the set of all buttons in the elevator problem)
  - pushed (the set of buttons that have been pushed)

Schema *Button_State*

Button_State

$$\begin{aligned} &\text{floor\_buttons}, \text{elevator\_buttons} : \mathbb{P}\ \text{Button} \\ &\text{buttons} : \mathbb{P}\ \text{Button} \\ &\text{pushed} : \mathbb{P}\ \text{Button} \\ \hline &\text{floor\_buttons} \cap \text{elevator\_buttons} = \varnothing \\ &\text{floor\_buttons} \cup \text{elevator\_buttons} = \text{buttons} \end{aligned}$$

Figure 12.26

3. Initial State

- The state when the system is first turned on

$$Button\_Init \mathrel{\hat{=}} [Button\_State' \mid pushed' = \varnothing]$$

(The caret ^ should be on top of the first equals sign =.
Unfortunately, this is hard to type in PowerPoint ☹)

4. Operations

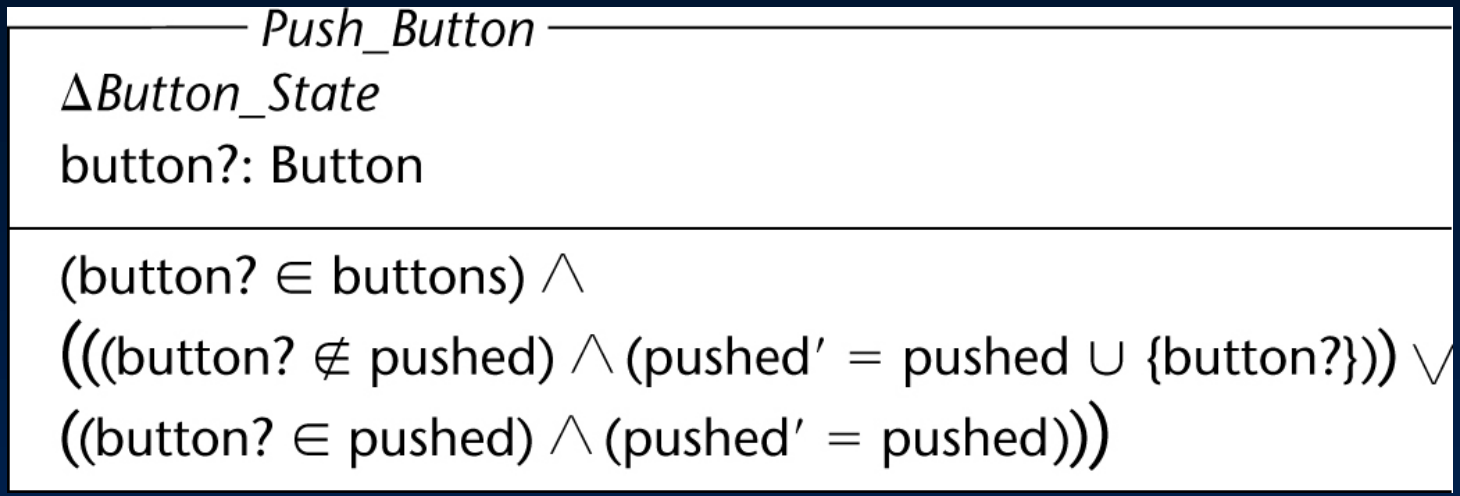

Figure 12.27

- A button pushed for the first time is turned on, and added to set *pushed*
- Without the third precondition, the results would be unspecified

Z: The Elevator Problem Case Study (contd)

Floor_Arrival

$\Delta Button\_State$

button?: Button

$$(button? \in buttons) \land$$
$$(((button? \in pushed) \land (pushed' = pushed \setminus \{button?\})) \lor$$
$$((button? \notin pushed) \land (pushed' = pushed)))$$

Figure 12.28

- If an elevator arrives at a floor, the corresponding button(s) must be turned off
- The solution does not distinguish between up and down floor buttons

12.9.2 Analysis of Z

- Z is the most widely used formal specification language

- It has been used to specify
  - CICS (part)
  - An oscilloscope
  - A CASE tool
  - Many large-scale projects (especially in Europe)

Analysis of Z (contd)

- Difficulties in using Z
  - The large and complex set of symbols
  - Training in mathematics is needed

Analysis of Z (contd)

- Reasons for the great success of Z
  - It is easy to find faults in Z specifications
  - The specifier must be extremely precise
  - We can prove correctness (we do not have to)
  - Only high-school math needed to *read* Z
  - Z decreases development time
  - A “translation” of a Z specification into English (or another natural language) is clearer than an informal specification

12.10 Other Formal Techniques

- Anna
  - For Ada
- Gist, Refine
  - Knowledge-based
- VDM
  - Uses denotational semantics ☹
- CSP
  - CSP specifications are executable
  - Like Z, CSP has a high squiggle factor

12.11 Comparison of Classical Analysis Techniques

- Formal methods are
  - Powerful, but
  - Difficult to learn and use
- Informal methods have
  - Little power, but are
  - Easy to learn and use
- There is therefore a trade-off
  - Ease of use versus power

Comparison of Classical Analysis Techniques (contd)

Classical Analysis Method	Category	Strengths	Weaknesses
Natural language (Section 12.2)	Informal	Easy to learn Easy to use Easy for the client to understand	Imprecise Specifications can be ambiguous, contradictory, or incomplete
Entity-relationship modeling (Section 12.6) PSL/PSA (Section 12.5) SADT (Section 12.5) SREM (Section 12.5) Structured systems analysis (Section 12.3)	Semiformal	Can be understood by the client More precise than informal techniques	Not as precise as formal techniques Generally cannot handle timing
Anna (Section 12.10) CSP (Section 12.10) Extended finite state machines (Section 12.7) Gist (Section 12.10) Petri nets (Section 12.8) VDM (Section 12.10) Z (Section 12.9)	Formal	Extremely precise Can reduce analysis faults Can reduce development cost and effort Can support correctness proving	Hard for the development team to learn Hard to use Almost impossible for most clients to understand

Figure 12.29

Newer Methods

- Many are untested in practice
- There are risks involved
  - Training costs
  - Adjustment from the classroom to an actual project
  - CASE tools may not work properly
- However, possible gains *may* be huge

Which Analysis Technique Should Be Used?

- It depends on the
  - Project
  - Development team
  - Management team
  - Myriad other factors
- It is unwise to ignore the latest developments

12.12 Testing during Classical Analysis

- Specification inspection
  - Aided by fault checklist
- Results of Doolan (1992)
  - 2 million lines of FORTRAN
  - 1 hour of inspecting saved 30 hours of execution-based testing

12.13 CASE Tools for Classical Analysis

- A graphical tool is exceedingly useful
- So is a data dictionary
  - Integrate them
- An analysis technique without CASE tools to support it will fail
  - The SREM experience

CASE Tools for Classical Analysis (contd)

- Typical tools
  - Analyst/Designer
  - Software through Pictures
  - System Architect

12.14 Metrics for CASE Tools

- Five fundamental metrics
- Quality
  - Fault statistics
  - The number, type of each fault
  - The rate of fault detection
- Metrics for “predicting” the size of a target product
  - Total number of items in the data dictionary
  - The number of items of each type
  - Processes vs. modules

12.15 Software Project Management Plan: The MSG Foundation Case Study

- The Software Project Management Plan is given in Appendix F

12.16 Challenges of Classical Analysis

- A specification document must be
  - Informal enough for the client; but
  - Formal enough for the development team
- Analysis (“what”) should not cross the boundary into design (“how”)
- Do not try to assign modules to process boxes of DFDs until the classical design phase

Overview of the MSG Foundation Case Study

Structured systems analysis	Section 12.4, Appendix D
Data flow diagram	Figure 12.9
Software project management plan	Section 12.15, Appendix F

Figure 12.30

Overview of the Elevator Problem Case Study

Requirements	Section 12.7.1
Finite state machine analysis	Section 12.7.1
Petri net analysis	Section 12.8.1
Z analysis	Section 12.9.1

Figure 12.31