Structured Systems Analysis
Three popular graphical specification methods of '70s

- DeMarco
- Gane and Sarsen
- Yourdon

All equivalent
All equally good
Many U.S. corporations use them for commercial products
Gane and Sarsen used for object-oriented design

Structured Systems Analysis Case Study

Sally's Software Store 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 Store 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?

Better question
What sections?
Still better
How? Batch, or online? In-house or out-service?

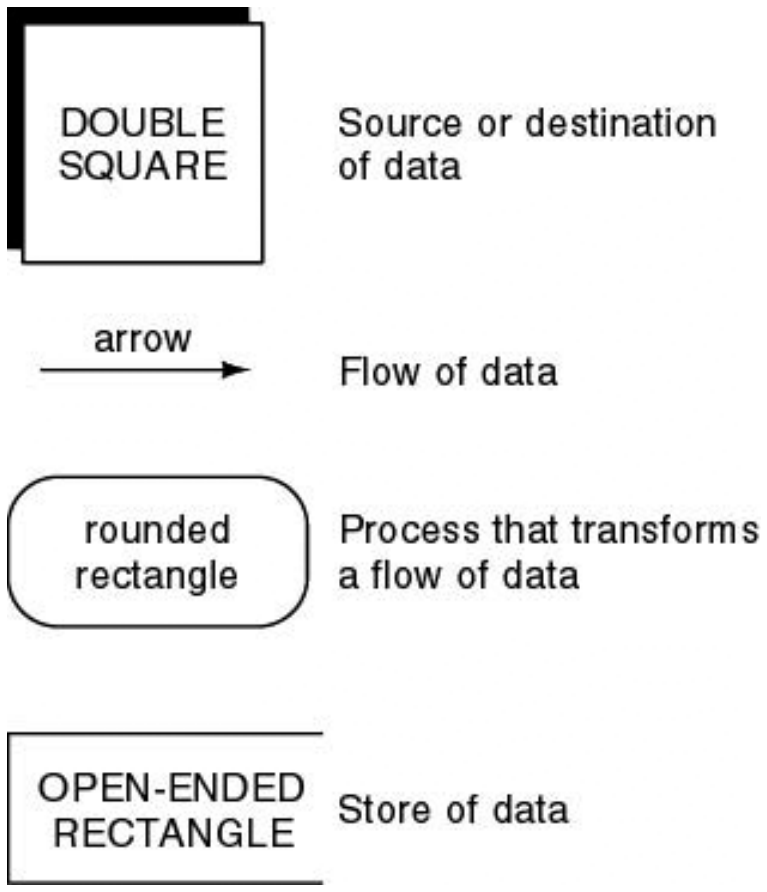
Case Study (contd)

- 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
- Assume: Computerization "in order to make more money"
  - Cost/benefit analysis for each section of business

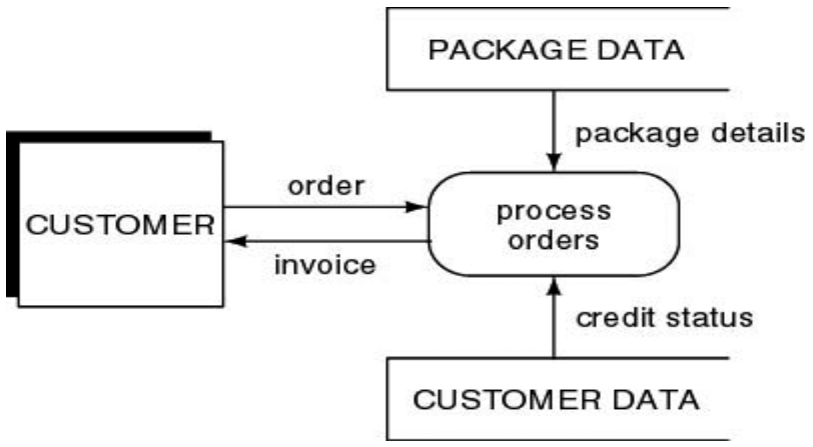
Case Study (contd)

- 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

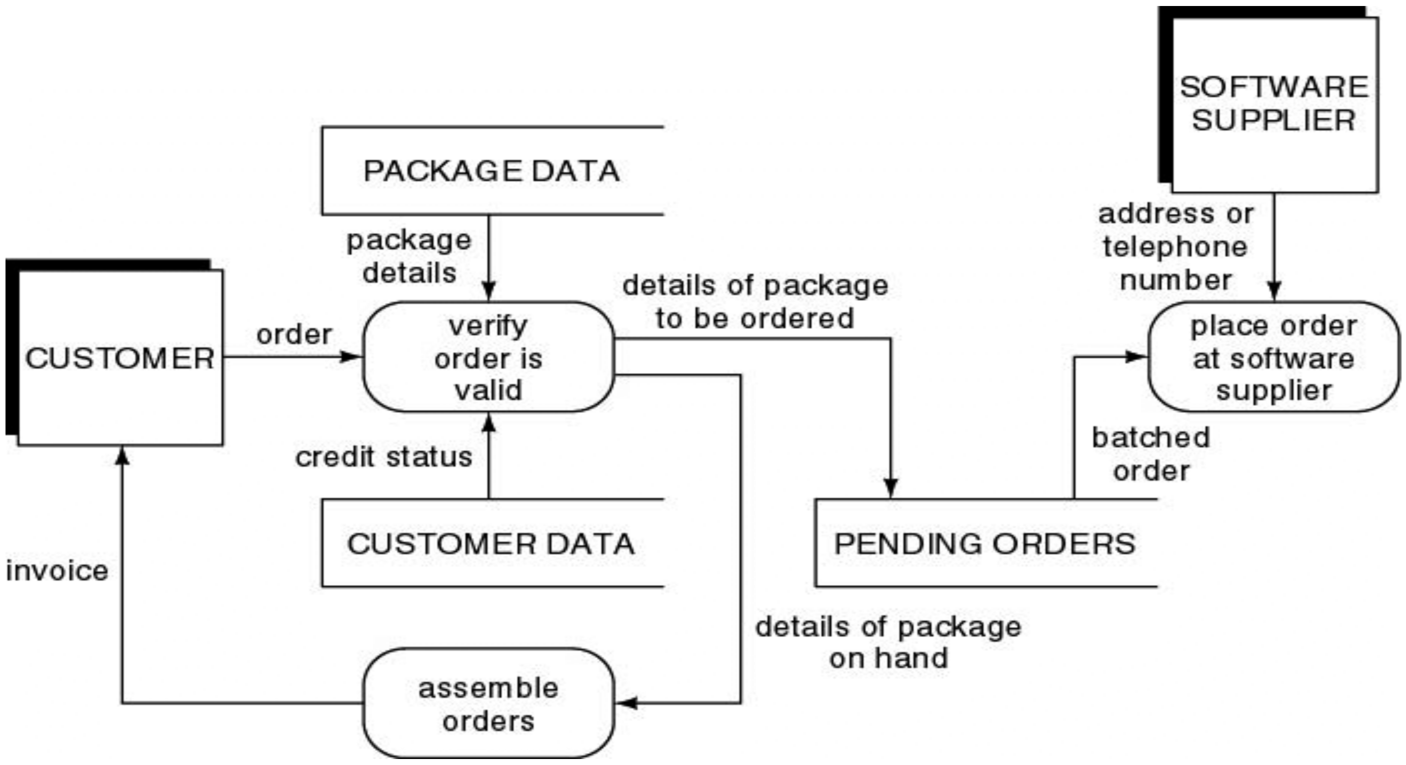
Case Study (contd)
Data flow diagram (DFD) shows logical data flow
"what happens, not how it happens"



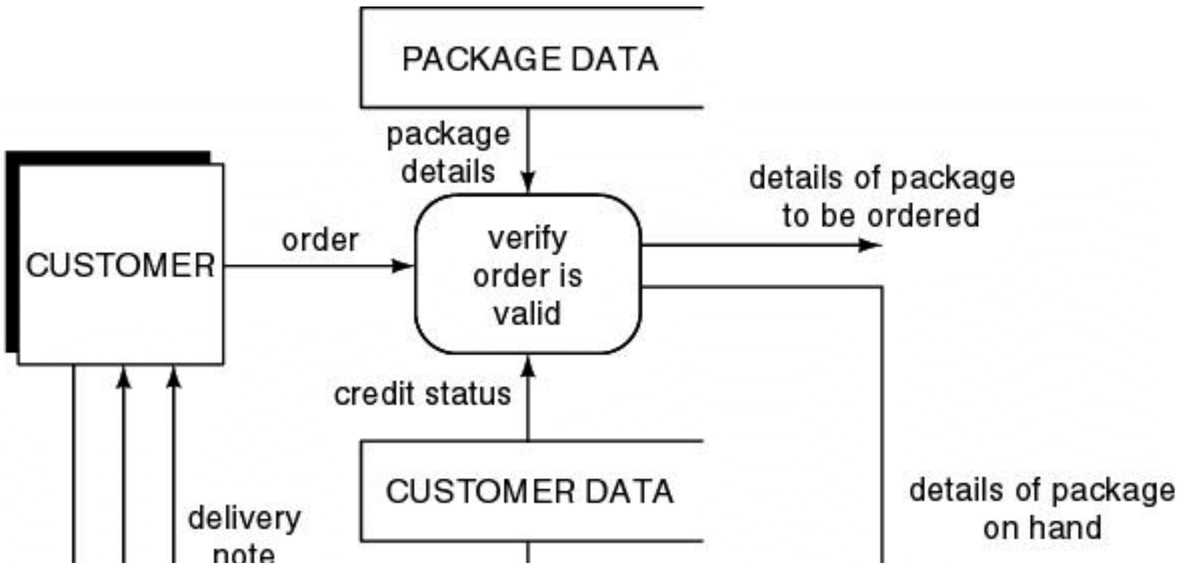
Step 1. Draw the DFD
First refinement
Infinite number of possible interpretations



Step 1 (contd)
Second refinement
pending orders scanned daily



Step 1 (contd)
Portion of third refinement



Step 2. Decide What Parts to Computerize

- Depends on how much client is prepared to spend
- Large volumes, tight controls
    - o Batch
- Small volumes, in-house microcomputer
    - o Online
- Cost/benefit analysis
- Returning to the example,
- one alternative is to automate accounts payable in batch and validate orders online.
- A second alternative is to automate everything

Step 3. Refine Data Flows

First, decide what data items must go into the various data flows. Then, refine each flow stepwise.

In the example, the data flow order can be refined as follows:

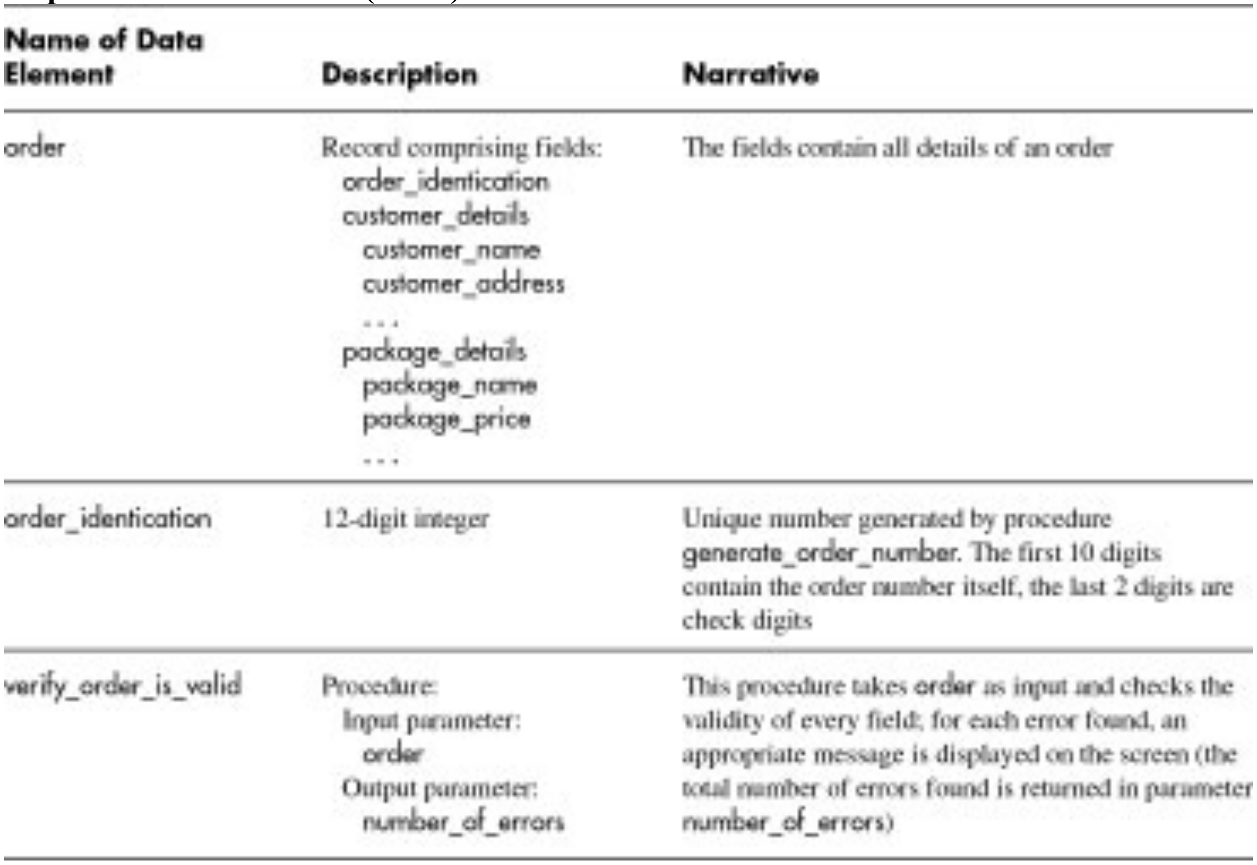
order:

- order_identifi cation
- customer_details
- package_details

Next, each of the preceding components of order is refined further.

In the case of a larger product, a data dictionary keeps track of all the data elements.

Step 3. Refine Data Flows (contd)



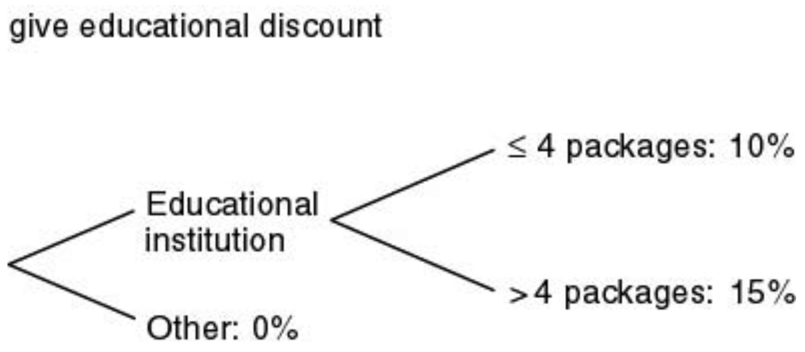
Name of Data Element	Description	Narrative
order	Record comprising fields: order_identication customer_details customer_name customer_address . . . package_details package_name package_price . . .	The fields contain all details of an order
order_identication	12-digit integer	Unique number generated by 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)

Sample data dictionary entries

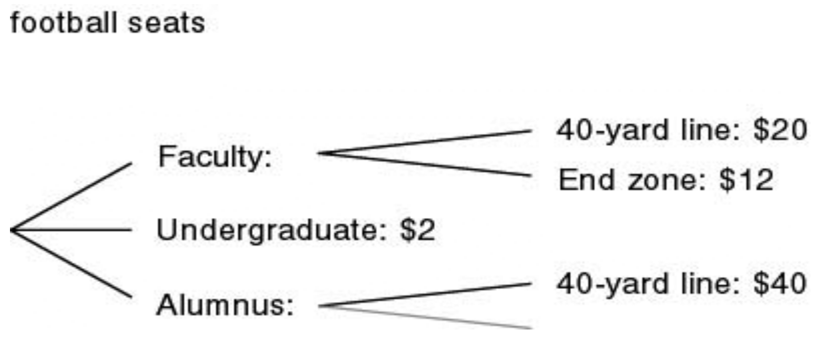
Step 4. Refine Logic of Processes
Determine what happens with in each process.
Have process give educational discount

- Sally must explain discount for educational institutions
- 10% on up to 4 packages, 15% on 5 or more

Translate into decision tree



Step 4 (contd)
Advantage of decision tree
Missing items are quickly apparent



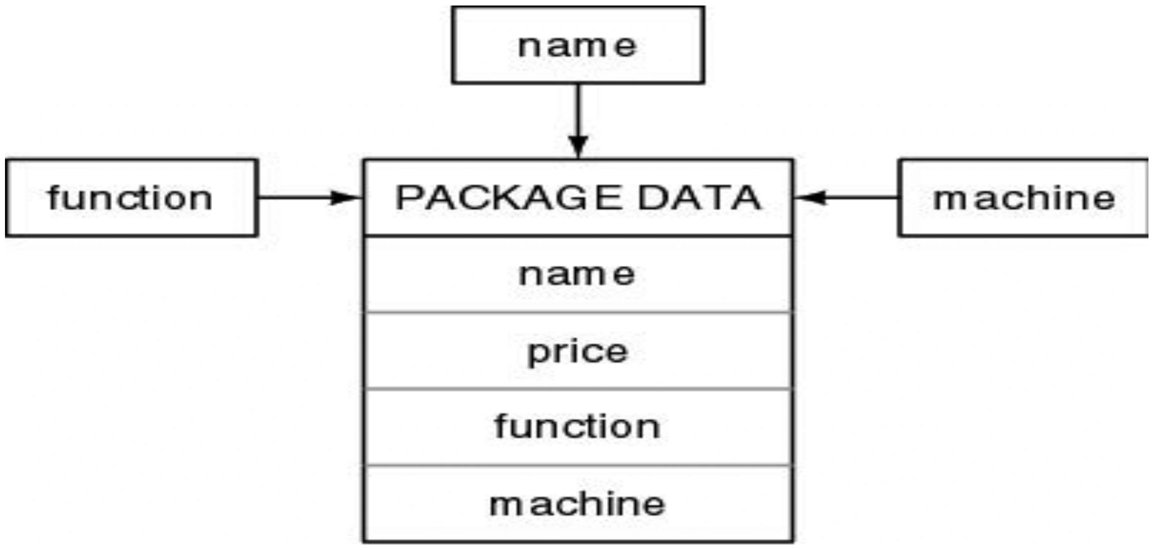
Can also use decision tables
CASE tools for automatic translation

Step 5. Define Data Stores
Define exact contents of each store and representation (format)

- COBOL: specify to pic level
- Ada: specify digits or delta

Specify where immediate access is required

- Data immediate access diagram (DIAD)



Step 6. Define Physical Resources
For each file, specify

- File name
- Organization (sequential, indexed, etc.)
- Storage medium
- Blocking factor(decisions can be made bcz of this)
- Records (to field level)

If a database management system(DBMS) is to be used, then the relevant information for each table is specifi ed here
Step 7. Determine Input/Output Specifications
Specify input forms, input screens, printed output
Step 8. Determine Sizing
Computing Numerical data for Step 9 to determine hardware requirements
Volume of input (daily or hourly)
Size, frequency of each printed report and its deadlines
Size, number of of each type passing between CPU and mass storage
Size of each file
Step 9. Hardware Requirements

- Mass storage for back-up
- Input needs
- Output devices
- Is existing hardware adequate?
- If not, recommend buy/lease

After approval by client: Specification document is handed to design team and software process continues

Petri Nets

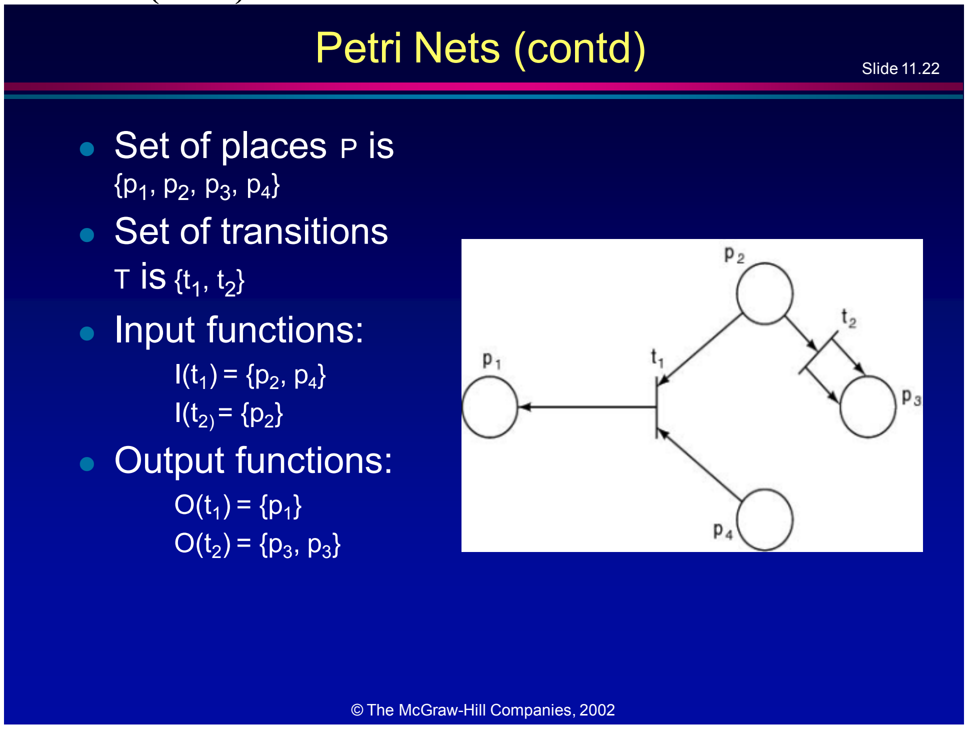
- A major difficulty with specifying real-time systems is coping with timing
    - This difficulty itself manifest in different ways
    - Synchronization problems
    - Race conditions
    - Deadlock

Often a consequence of poor specifications

Petri Nets (contd)

- Petri nets
    - Powerful technique for specifying systems with potential timing problems
- A Petri net consists of four parts:
    - Set of places P
    - Set of transitions T
    - Input function I
    - Output function O

Petri Nets (contd)



Petri Nets (contd)

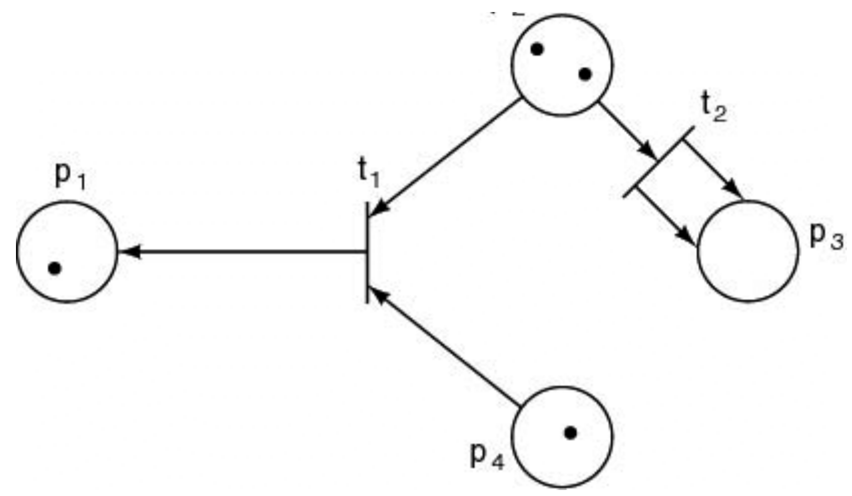
Slide 11.22

- 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:
    $O(t_1) = \{p_1\}$
    $O(t_2) = \{p_3, p_3\}$

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 ® P咲 is *input* function, mapping from transitions to bags of places
- : T ® P咲 is *output* function, mapping from transitions to bags of places
- (A *bag* is a generalization of sets which allows for multiple instances of element in bag, as in example above)
- *Marking* of a Petri net is an assignment of tokens to that Petri net

Petri Nets (contd)



Four tokens, one in p_1, two in p_2, none in p_3, and one in p_4

Represented by vector (1,2,0,1)

Transition is enabled if each of its input places has as many tokens in it as there 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

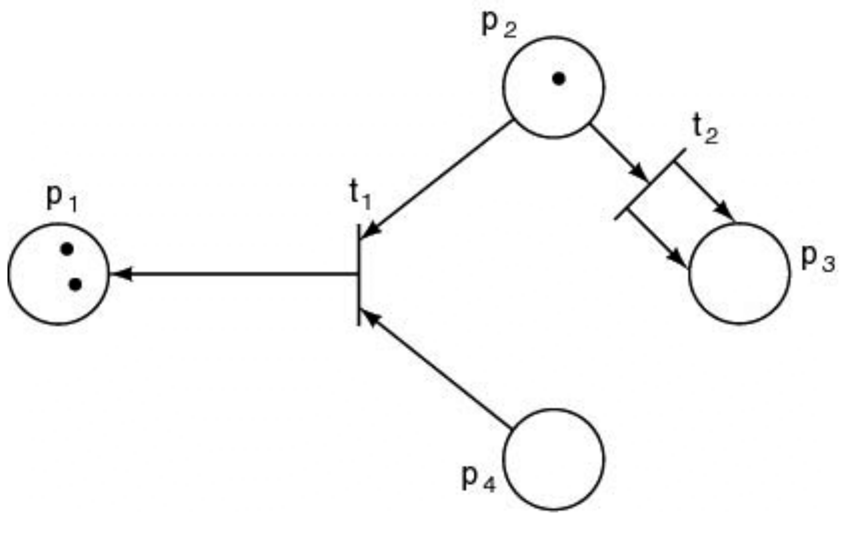
Important: Number of tokens is not conserved

Transition t_2 is also enabled

Petri Nets (contd)

Petri nets are indeterminate

Suppose t_1 fires

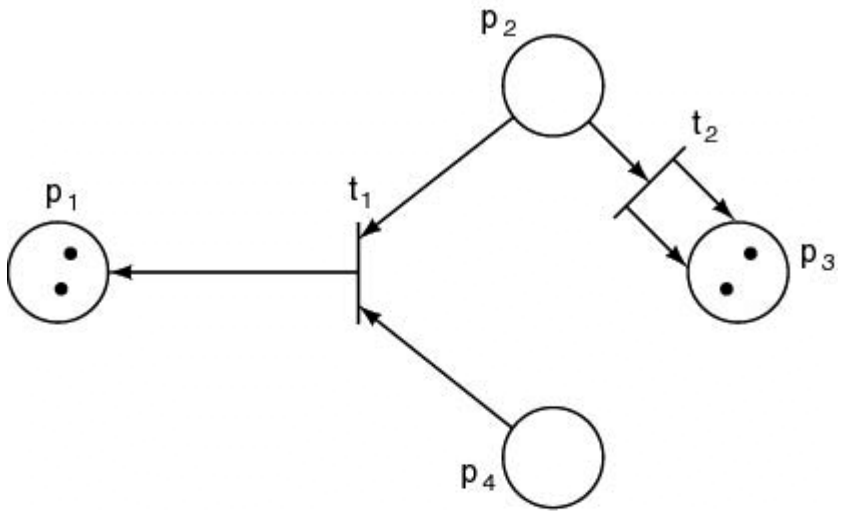


Resulting marking is (2,1,0,0)

Petri Nets (contd)

Now only t_2 is enabled

It fires



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 N

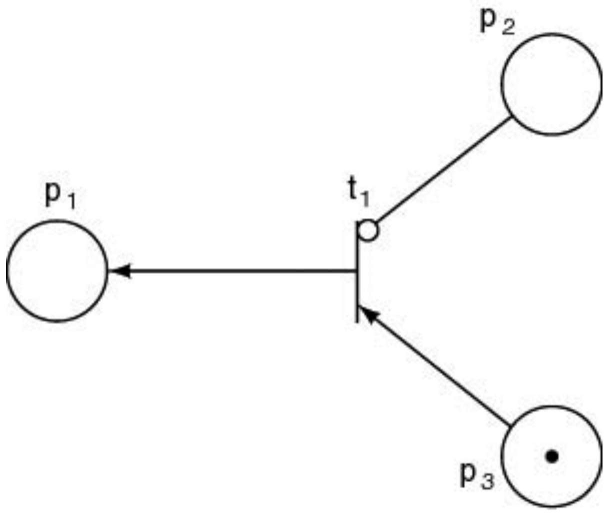
M : P ® N

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

Petri Nets (contd)

Inhibitor arcs

Inhibitor arc is marked by small circle, not arrowhead



Transition t_1 is enabled

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

Elevator Problem: Petri Net

- Product is to be installed to control n elevators in a building with m floors

- Each floor represented by place F_f, $1 \leq f \leq m$
- Elevator represented by token
  . Token in F_f denotes that an elevator is at floor F_f

Elevator Problem: Petri Net (contd)

First constraint

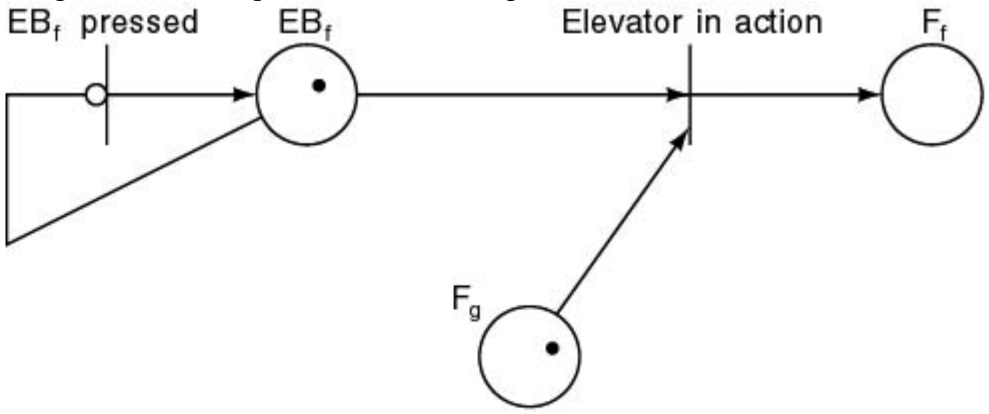
1. Each elevator has a set of m buttons, one for each floor. These illuminate when pressed and cause the elevator to visit the corresponding floor. The illumination is canceled when the corresponding floor is visited by an elevator

Elevator button for floor f is represented by place EB_f, $1 \leq f \leq m$

Token in EB_f denotes that the elevator button for floor f is illuminated

Elevator Problem: Petri Net (contd)

A button must be illuminated the first time button is pressed and subsequent button presses must be ignored .this is specified in below fig



If button EB_f is not illuminated, no token in place and transition EB_f pressed is enabled

Transition fires, new token is placed in EB_f

Now, no matter how many times button is pressed, transition EB_f pressed cannot be enabled

Elevator Problem: Petri Net (contd)

When elevator reaches floor g, token is in place F_g, transition Elevator in action is enabled, and then fires

Tokens in EB_f and F_g removed

This turns off light in button EB_f

New token appears in F_f

This brings elevator from floor g to floor f

Elevator Problem: Petri Net (contd)

Motion from floor g to floor f cannot take place instantaneously

Timed Petri nets

Elevator Problem: Petri Net (contd)

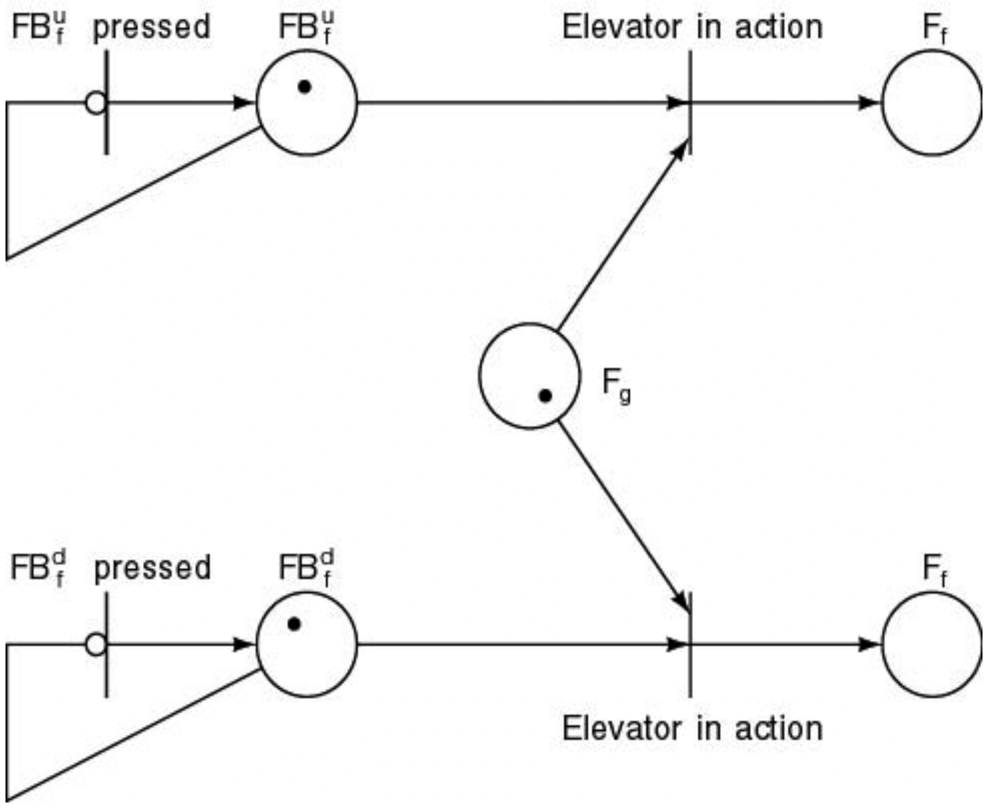
Second constraint

2. Each floor, except the first and the top floor, has 2 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 represented by places FB^u_f and FB^d_f

Elevator Problem: Petri Net (contd)



Elevator Problem: Petri Net (contd)

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)

Elevator Problem: Petri Net (contd)

Third constraint

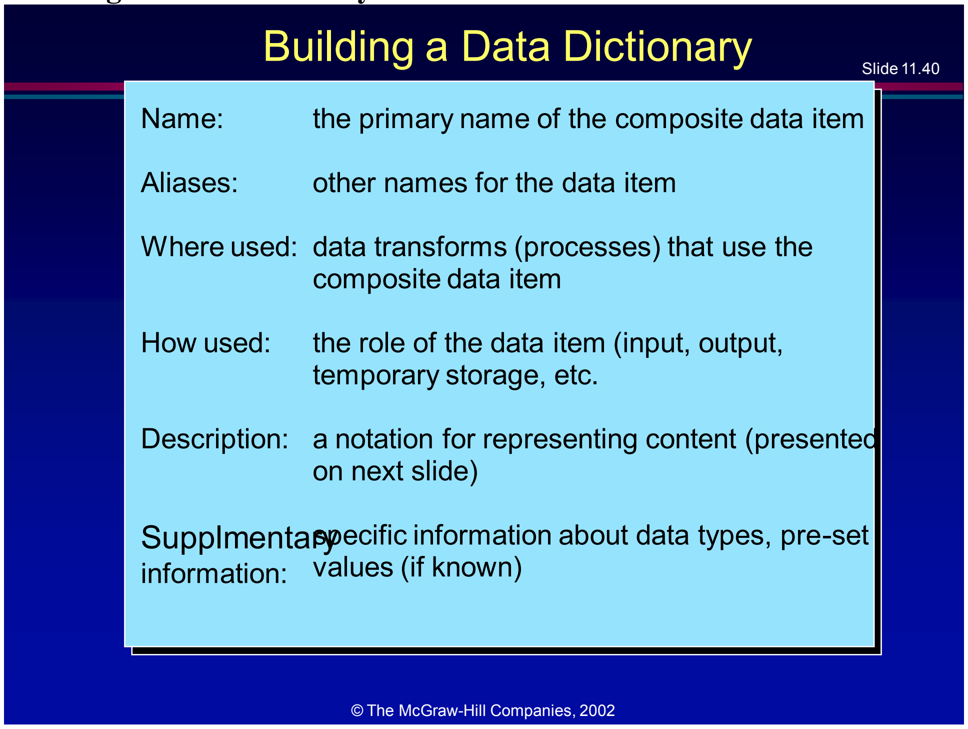
C_3. If an elevator has no requests, it remains at its current floor with its doors closed

If no requests, no Elevator in action transition is enable

The Data Dictionary

- ❑ a quasi-formal grammar for describing the content of data that the software will process and create
- ❑ a notation for describing control data and the values that control data can take, e.g., "on," or "off"
- ❑ a repository that also contains "where-used" / "how used" information
- ❑ a notation that can be represented manually, but is best developed using CASE tools

Building a Data Dictionary



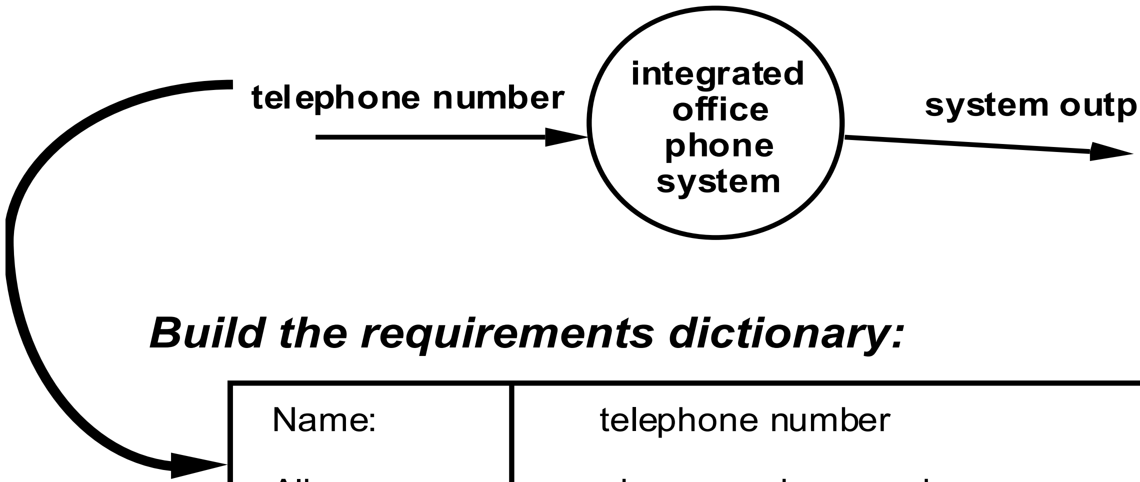
Building a Data Dictionary

Name:	the primary name of the composite data item
Aliases:	other names for the data item
Where used:	data transforms (processes) that use the composite data item
How used:	the role of the data item (input, output, temporary storage, etc.
Description:	a notation for representing content (presented on next slide)
Supplmentary information:	specific information about data types, pre-set values (if known)

Data Dictionary Notation

Notation	*Meaning*
=	is composed of
+	and
[|]	either-or
$\{\quad\}^n$	n repetitions of
(...)	optional data
* ... text ...*	delimits a comment

Data Dictionary Example



Build the requirements dictionary:

Name:	telephone number
Aliases:	phone number, number
Where/How used:	read-phone-number (input) display-phone-number (output) analyze-long-distance-calls (input)
Description:	telephone no. = [local extension | outside no. | 0] outside no. = 9 + [service code | domestic no.] service code = [211 | 411 | 611 | 911] domestic no. = ((0) + area code) + local number area code = *three numeral designator*
Format:	alphanumeric data