I226
Computer Networks

Chapter 11

Design of Network Equipments and Protocols

Assoc. Prof. Yuto Lim

School of Information Science
Japan Advanced Institute of Science and Technology
Nomi, Ishikawa, Japan

<u>Objectives of this Chapter</u>

- Give an understanding what is the <u>protocol design</u> and <u>protocol engineering</u>
- Explain the <u>FSM model</u> that used for network protocol design
- Describe a few examples of <u>specification language</u> for protocol and software designs
- Provide the knowledge and tools of the <u>computer network design</u> and its network examples

Outline

- Protocol Design
  - Behavioral Model – Finite State Machine (FSM)
  - Specification Languages
    - State Charts (SC)
    - Specification and Description Language (SDL)
    - Unified Modeling Language (UML)
    - Message Sequence Charts (MSC)
    - Abstract Syntax Notation One (ASN.1)
  - Testing Language
    - Tree and Tabular Combined Notation (TTCN)
- Computer Network Design

Protocol Design

- Implementing a protocol is only reasonably difficult
- Designing a good protocol is more difficult
- Design starts from the requirements:
  - Define the problem to be solved
  - Define why no existing protocol is good enough
- It is a good idea to supplement the requirements with practical usage cases
- Protocol design tools exist and should be used
  - Help to avoid some pitfalls, like deadlocks

Protocol Engineering

- Covers the whole life-cycle of the protocol
  - Requirements specification
  - Protocol design and specification
  - Specification verification
  - Protocol implementation
  - Implementation testing

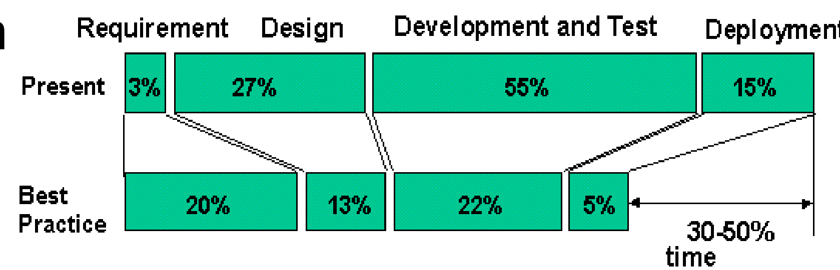
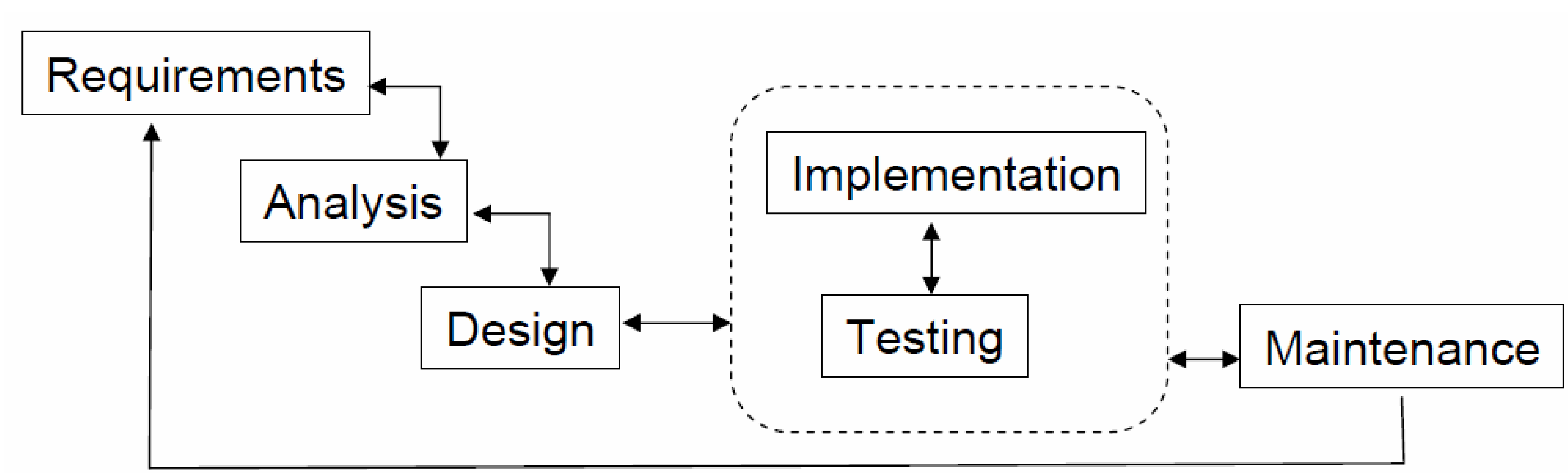


Fig. Reduced time to market (as a product)

- Often include heavy iteration



Protocol Engineering as a Discipline

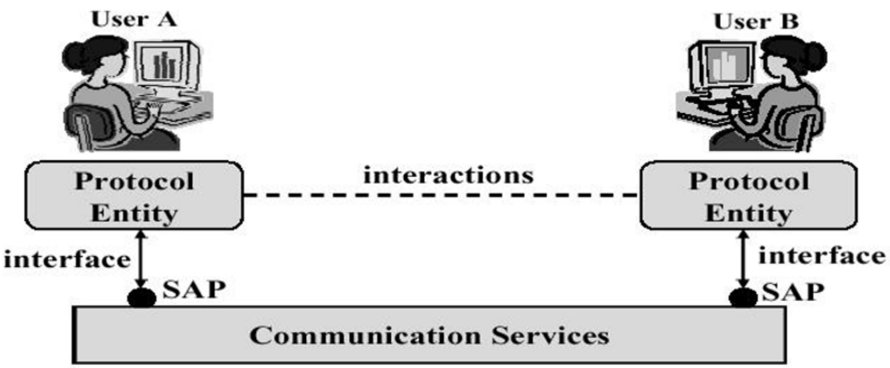
- Combines
  - Communication systems
  - Formal languages
  - Software engineering
- These skills are needed to produce protocols that
  - Are unambiguous and error free
  - Meet practical environmental constraints
  - Meet communications requirements

Protocol Architectures, Designing Layers

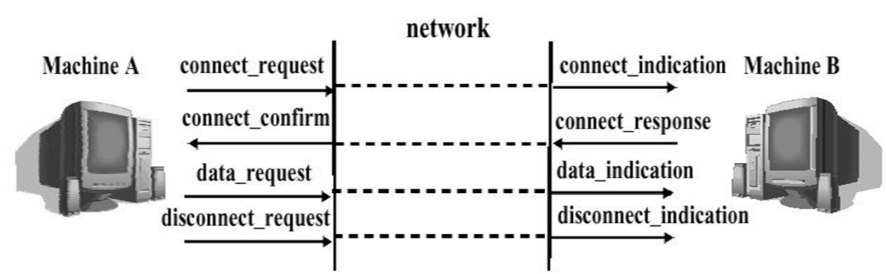
- Allocate a well-defined function for each layer
- Create a layer boundary at a point where the no. of interactions is minimized
- Design interlayer interactions as service primitives
- Keep the no. of layers to minimum
- Use layers to allow different levels of abstraction
- Allow changes to layer functions
  - Even changing a protocol without affecting other layers
- Set a functional protocol entities composing a stack

Protocol Specification

- Components of a protocol:
  - Communication services
  - Entities
  - interfaces
  - Interactions

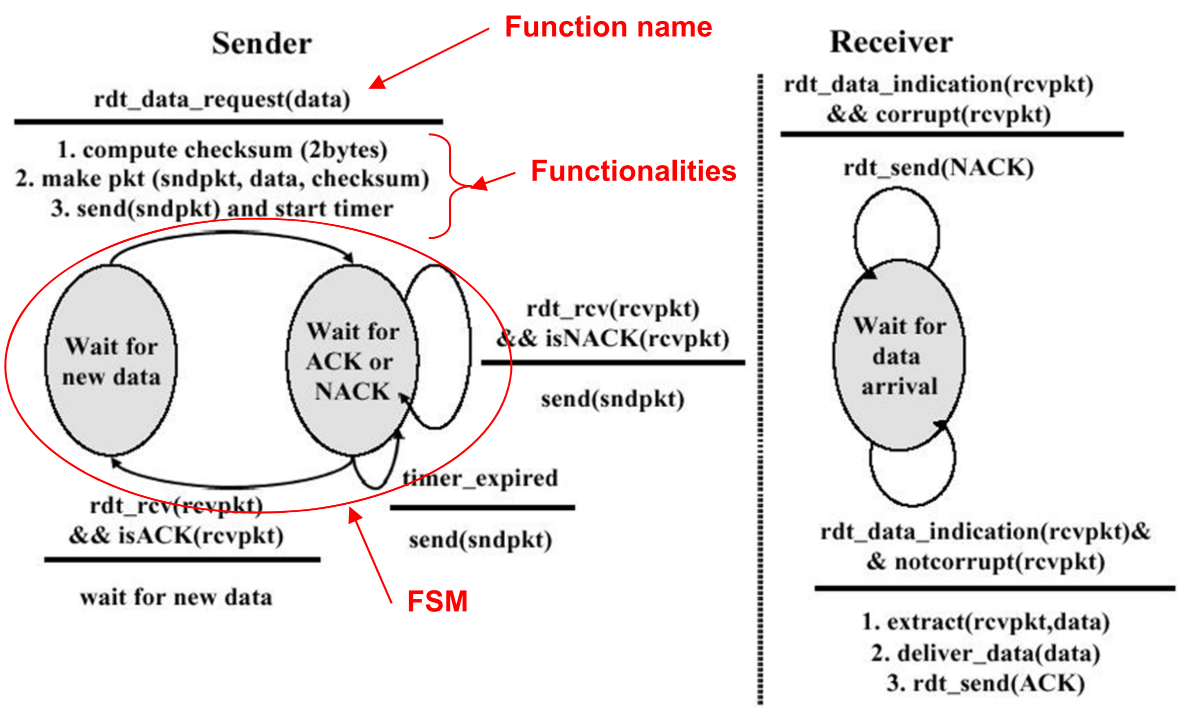


- Specification of protocol design:
  - Allows the designer to prepare an abstract model of the protocol for testing and analysis
- Specification of communication service:
  - Service primitives – request, response, indication, and confirm
  - Service parameters – data size, checksum size, caller address, etc.



Example: Reliable Data Transfer

- Reliable data transfer specification using finite state machine (FSM) or finite state automaton (FSA)




```
<Sender> rdt_data_request(data):
        compute checksum, make_pkt, send(sndpkt)
<Sender> rdt_rcv(rcvpkt):
        performs reception of packet (ACK or
        NACK). If ACK then send next data, if
        NACK then retransmit the data
<Sender> isNACK(rcvpkt):
        checks for negative acknowledgment
        reception
<Sender> isACK(rcvpkt):
        checks for positive acknowledgment
        reception
<Sender> send(sndpkt):
        sends the packet

<Receiver> rdt_data_indication(rcvpkt):
        receives the data and sends an ACK, In
        case of corrupted data reception, sends
        NACK
<Receiver> rdt_send(ACK):
        sends the ACK
<Receiver> rdt_send(NACK):
        sends the NACK
<Receiver> corrupt(rcvpkt):
        checks the packet for corrupted data
<Receiver> notcorrupt(rcvpkt):
        checks for non corruption
<Receiver> extract(rcvpkt,data):
        extracts the data from received packet
<Receiver> deliver data(data):
        delivers data to the upper layer
```

FSM Model

- FSM is an abstract representation of behavior exhibited by some systems

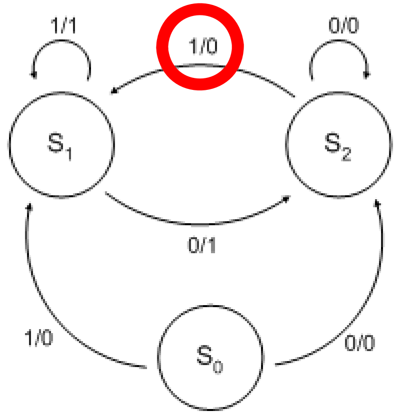
- FSM can serve any of two roles:
  - As a specification of the required behavior, and/or
  - As a design artifact (object) according to which an application is to be implemented

- FSM is different from State Charts
  - While FSM can be modeled using state charts, the reverse is not true

- Term of state diagram is often used to denote a graphical representation of an FSM

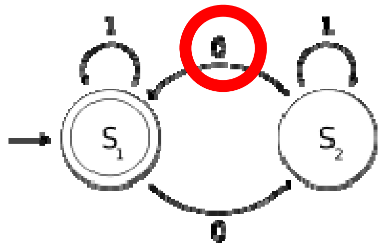
Formal Definition of FSM

- FSM (Mealy-type) is a 6-tuple $F<S, I, O, F, H, s_0>$
  - G.H. Mealy, 1955 publication
    - Associates outputs with transitions (H maps $S \times I \rightarrow O$)
  - S is a set of all states $\{s_0, s_1, \ldots, s_l\}$
  - I is a set of inputs $\{i_0, i_1, \ldots, i_m\}$
  - O is a set of outputs $\{o_0, o_1, \ldots, o_n\}$
  - F is a next-state function ($S \times I \rightarrow S$)
  - H is an output function ($S \times I \rightarrow O$)
  - s_0 is an initial state



"i/o" at edge shows input/output

- FSM (Moore-type) is a 7-tuple $F<S, I, O, F, H, s_0, E>$
  - E.F. Moore, 1956 publication
    - Associates outputs with states (H maps $S \rightarrow O$)
  - H is an output function ($S \rightarrow O$)
  - E is the set of final or accepting or terminating states



"i" at edge shows input only

- Shorthand notations to simplify descriptions
  - Implicitly assign 0 to all unassigned outputs in a state
  - Implicitly AND every transition condition with clock edge (FSM is synchronous)

Specification Language (SL)

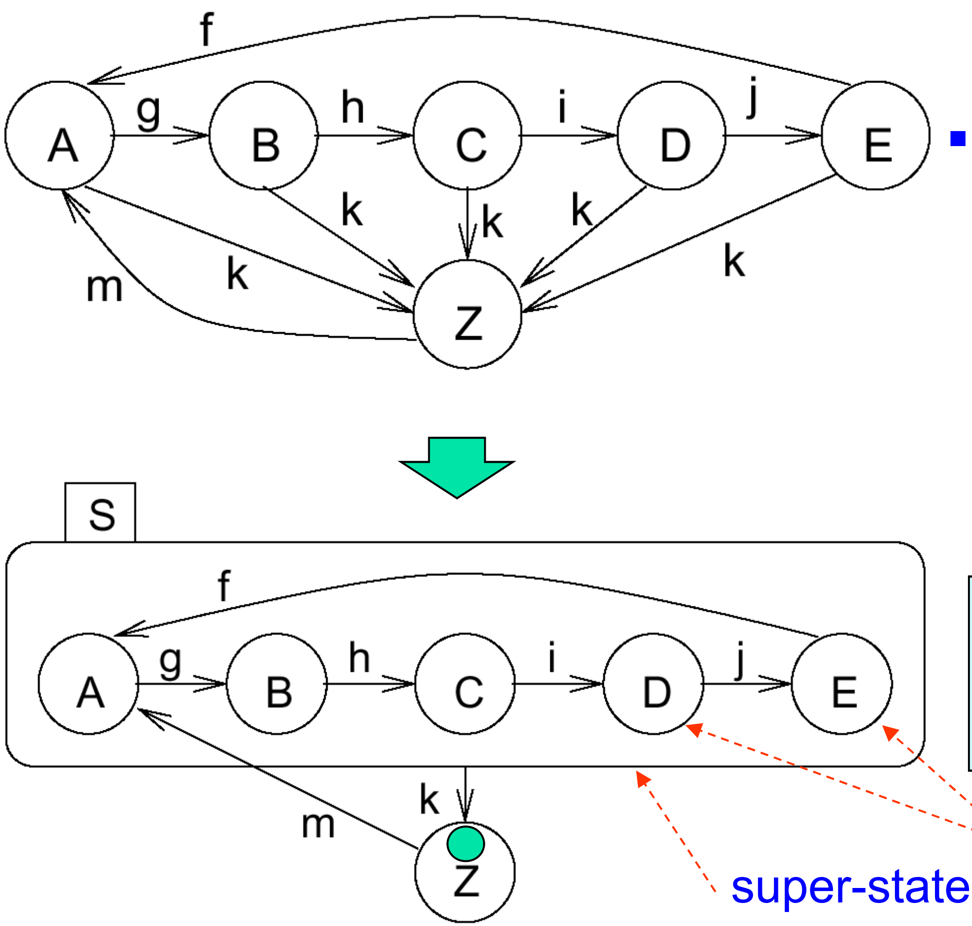
- Provide a clear description of the system functionality
  - Definition: SL is a formal language in computer science used during systems analysis, requirements analysis and systems design to describe a system at a much higher level than a programming language, which is used to produce the executable code for a system (quoted from Wikipedia)
- Support different models of computations
  - FSM model, data flow model
- Should not limit to implementation options
- Enable computer-aided analysis techniques
  - Estimation
  - Exploration
  - Partitioning
  - Synthesis
  - Validation
  - Testing

Protocol Design Requirements using SL

- Executability
  - Validation through simulation
- Synthesizability
  - Implementation in HW and/or SW
  - Support for IP reuse
- Modularity
  - Hierarchical composition
- Event-handling
  - External or internal events
- Domain-specific support
- Simplicity

- Portability and flexibility
- Completeness
  - Support for all concepts found in embedded system
    - concurrency, communication, synchronization, etc
- Orthogonality
  - Orthogonal constructs for orthogonal concepts
    - communication vs. computation
- Termination
  - It should be clear, at which time all computations are completed

State Charts (SC)



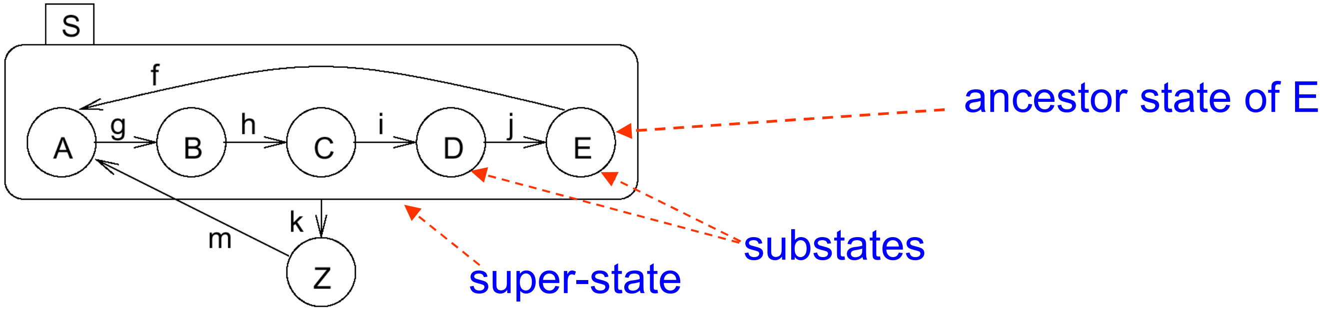
- Motivation on deficits of FSM for modeling
  - Only one sequential process, no concurrency
  - No hierarchical structuring capabilities
- Improvement:
  - State Charts (SC) is proposed by D. Harel, 1987
  - SC introduces hierarchy, concurrent, and computation
  - SC is used in many tools for the specification, analysis and simulation of discrete event systems, e.g., MATLAB-StateFlow, UML, Rhapsody, Magnum
  - Complicated semantics

FSM will be in exactly one of the substates of S if S is active (either in A or in B or etc...)

Concurrency

Definitions

- Active states: current states of FSM
- Basic states: states not composed of other states
- Super-states: states containing other states
- For each basic state s, the super-states containing s are called ancestor states
- Super-states S are called OR-super-states, if exactly one of the substates is active whenever S is active



State Charts Discussion

- Pros:
  - Hierarchy allows nesting of AND/OR-super-states
  - Large number of commercial simulation tools available (StateMate, StateFlow from MATLAB, BetterState, etc)
  - Back-end tools translate State Charts into C or VHDL – enabling HW/SW implementations
- Cons:
  - Generated C programs frequently inefficient
  - Not useful for distributed applications
  - No program constructs directly
  - No object-oriented support

⇒ We need a new specification language, e.g., SDL

SDL

- Specification and description language (SDL) is an object-oriented, formal language
    - Designed for specification of distributed systems
    - http://www.sdl-forum.org/

- Defined by ITU: Z.100 recommendation in 1980

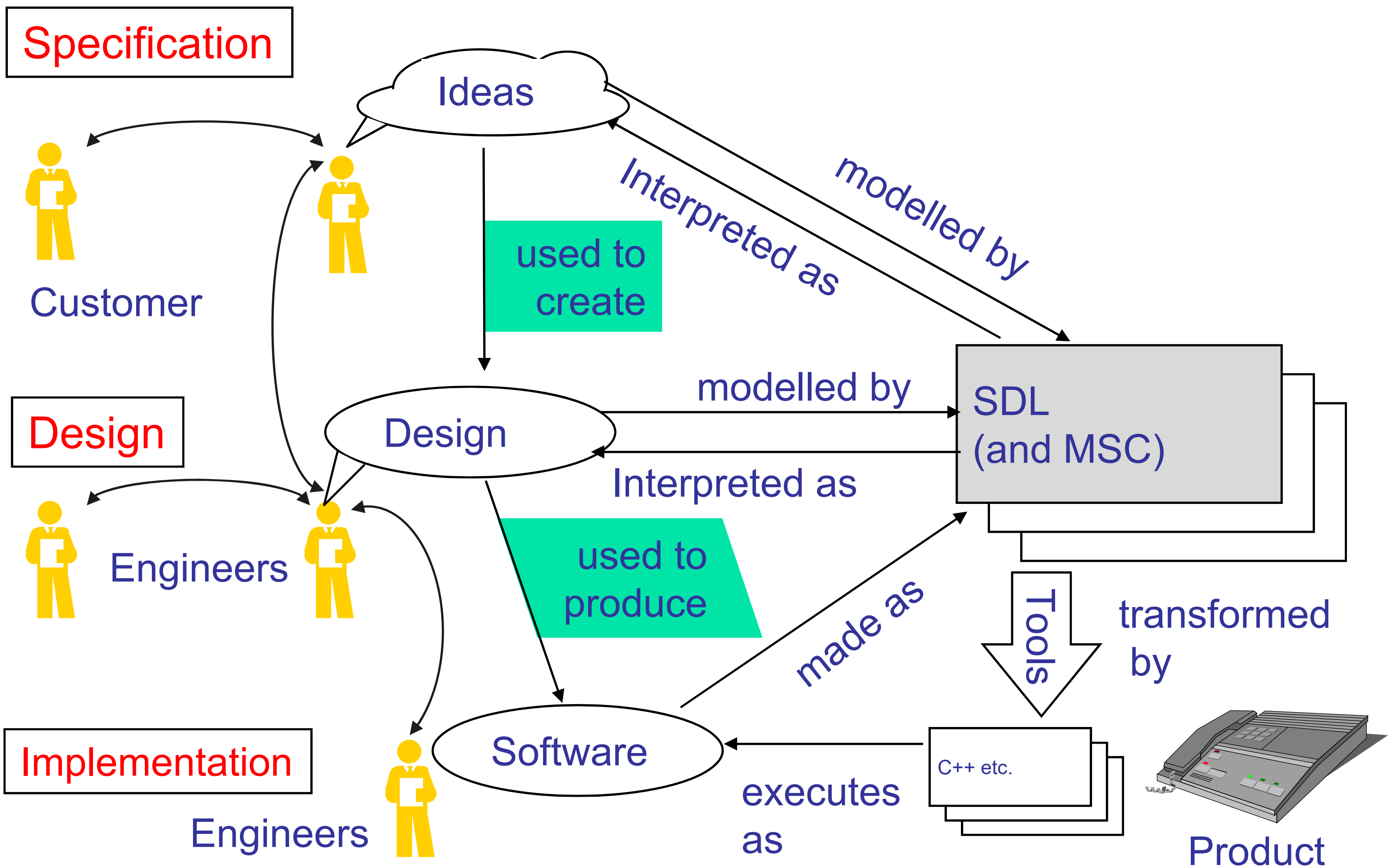
- Describe the architecture, behavior and data of distributed systems in real-time environments

- Provides textual and graphical formats

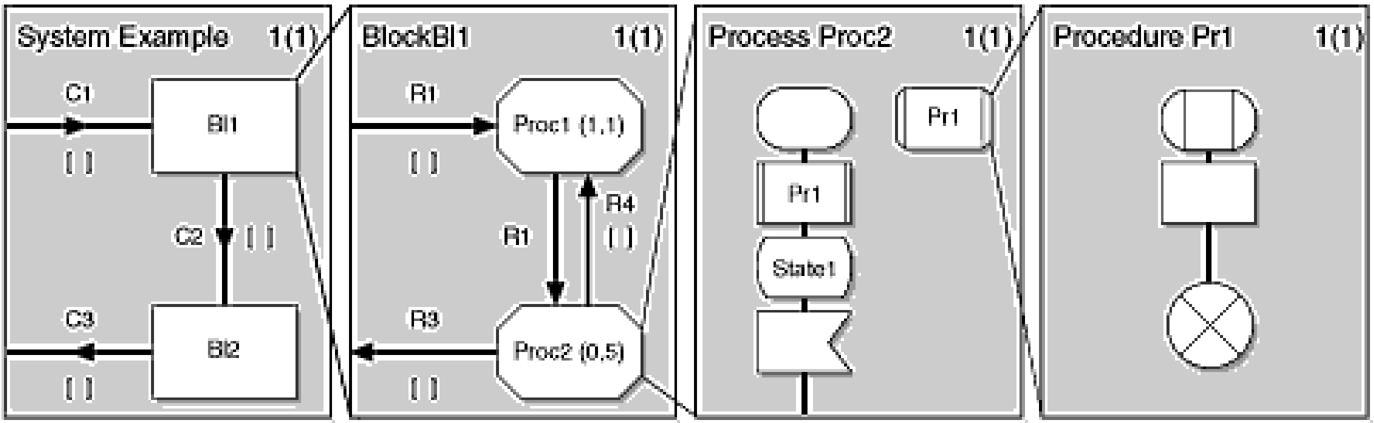
- Like State Charts, it is based on communicating FSM (CFSM) model of computation; each FSM is called a process

- It uses message passing for communications, and supports operations on data

Use of SDL

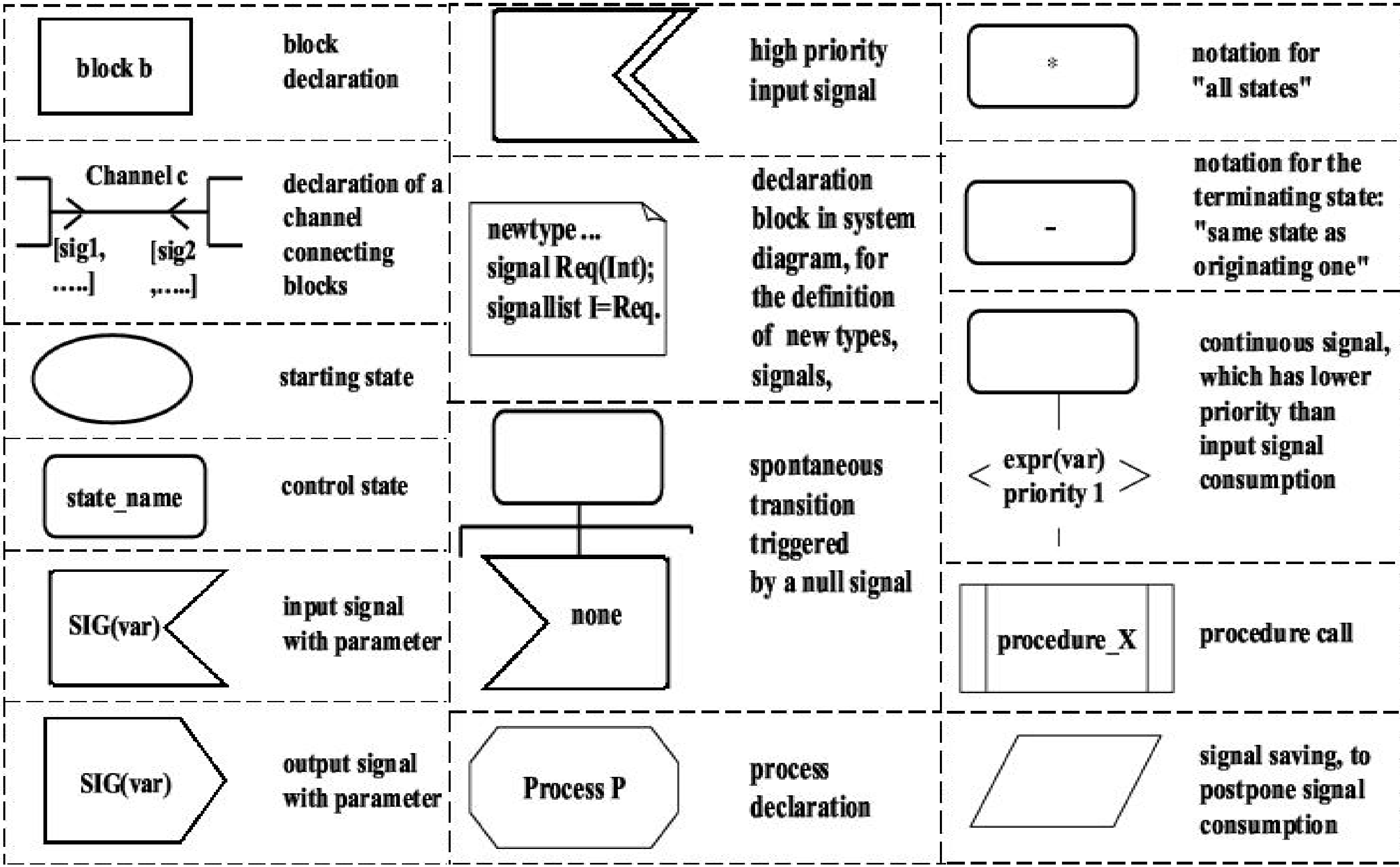

Specification
Ideas
Customer
used to create
Interpreted as
modelled by
Design
Design
modelled by
SDL
(and MSC)
Interpreted as
Engineers
used to produce
made as
Tools
transformed by
Implementation
Software
C++ etc.
Engineers
executes as
Product

SDL Structure

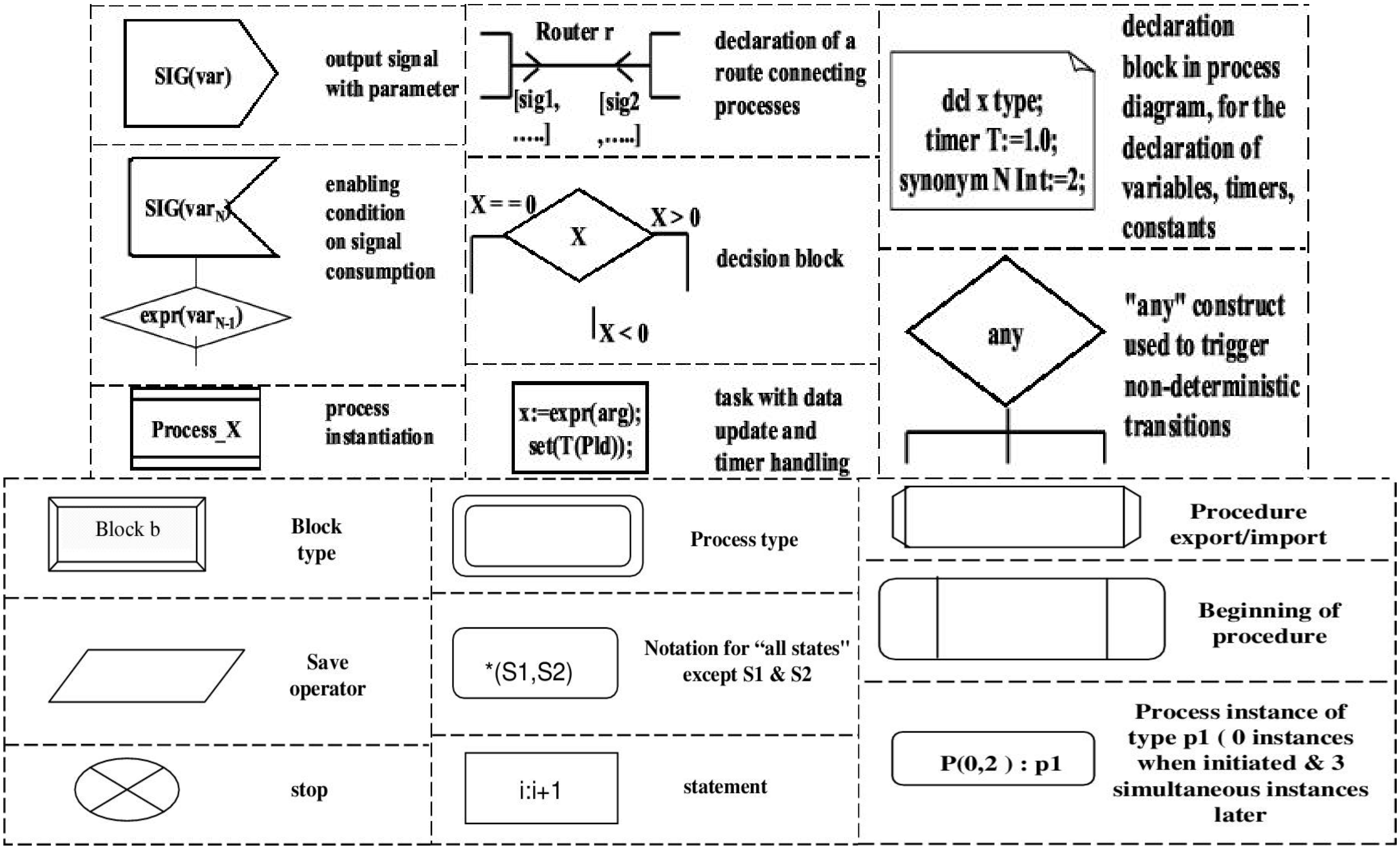


- Comprises 4 main hierarchical levels:
  - System
  - Block
  - Process
  - Procedure
- System contains one or more blocks, interconnected with each other and with the boundary of the system by channels
- Block can be partitioned into sub-blocks and channels
- Channel is a means of conveying signals

SDL Syntax

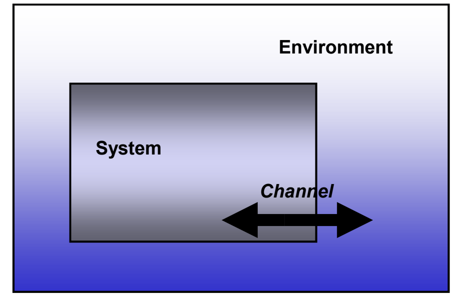

block b
block declaration
Channel c
[sig1, ...]
[sig2 ,.....]
declaration of a channel connecting blocks
starting state
state_name
control state
SIG(var)
input signal with parameter
SIG(var)
output signal with parameter
high priority input signal
newtype ...
signal Req(Int);
signallist I=Req.
declaration block in system diagram, for the definition of new types, signals,
none
spontaneous transition triggered by a null signal
Process P
process declaration
*
notation for "all states"
-
notation for the terminating state: "same state as originating one"
expr(var) priority 1
continuous signal, which has lower priority than input signal consumption
procedure_X
procedure call
signal saving, to postpone signal consumption

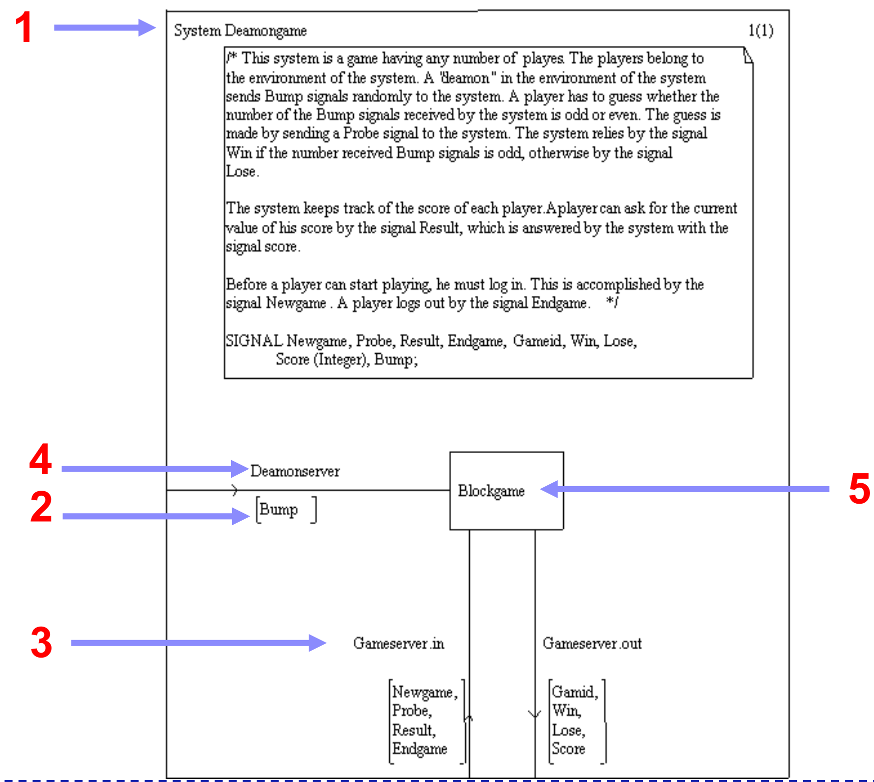
SDL Syntax (cont.)



System

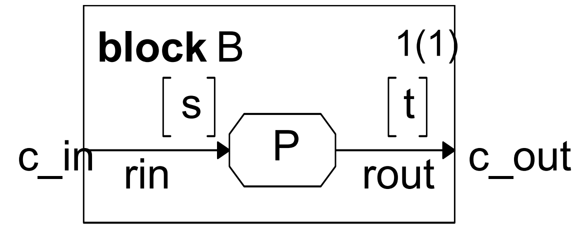
- System description constitutes the top level of detail
- System is what the SDL description specifies:
  - An abstract machine communicating with its environment
- System diagram usually contains the following elements:
  1. System name
  2. Signal descriptions
  3. Channel descriptions
  4. Data type descriptions
  5. Block descriptions

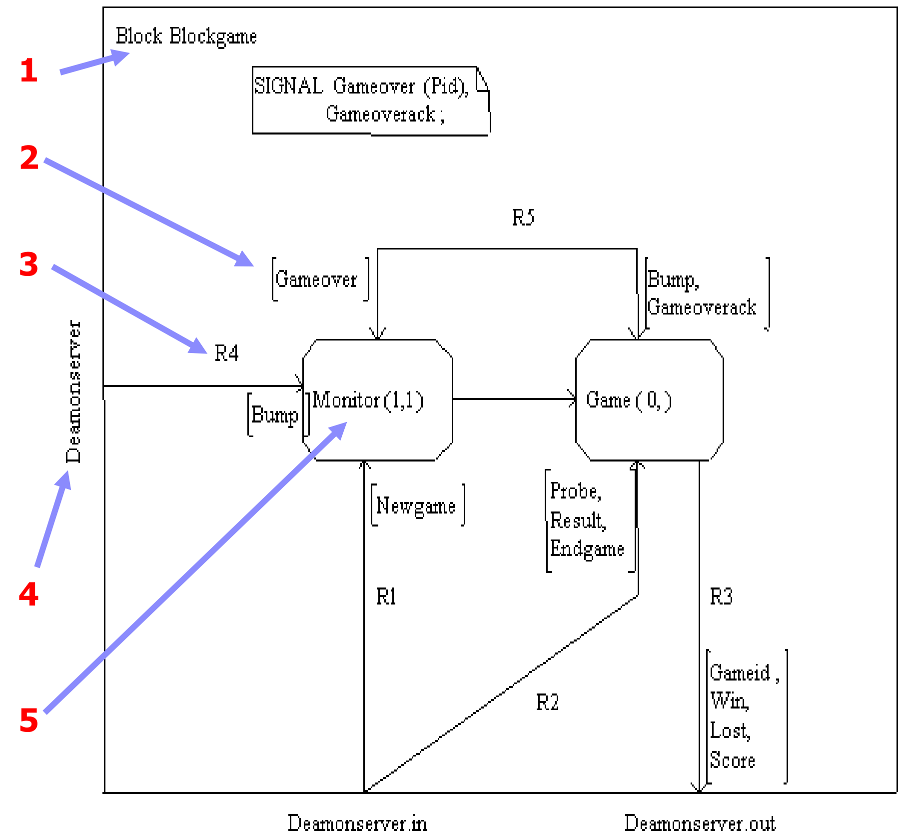




Block

- Block is a part of the system that can be treated as a self-contained object
- Block diagram usually contains the following elements:
  1. Block name
  2. Signal descriptions
  3. Signal route descriptions
  4. Channel-to-route connections
  5. Process descriptions





“Sketch Panel” to “Block Panel”

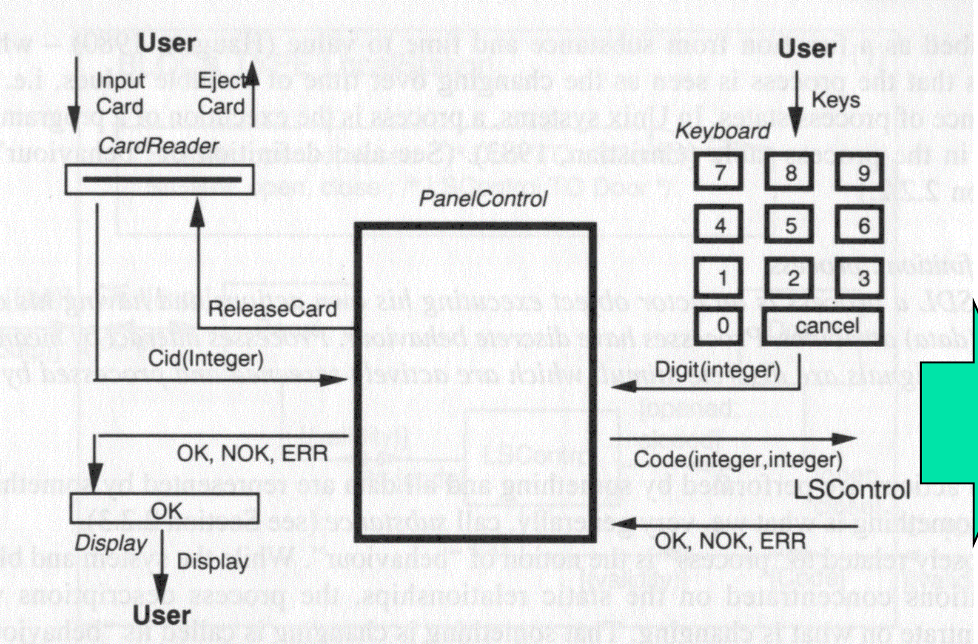


Figure 4.11 *Sketch of Panel*

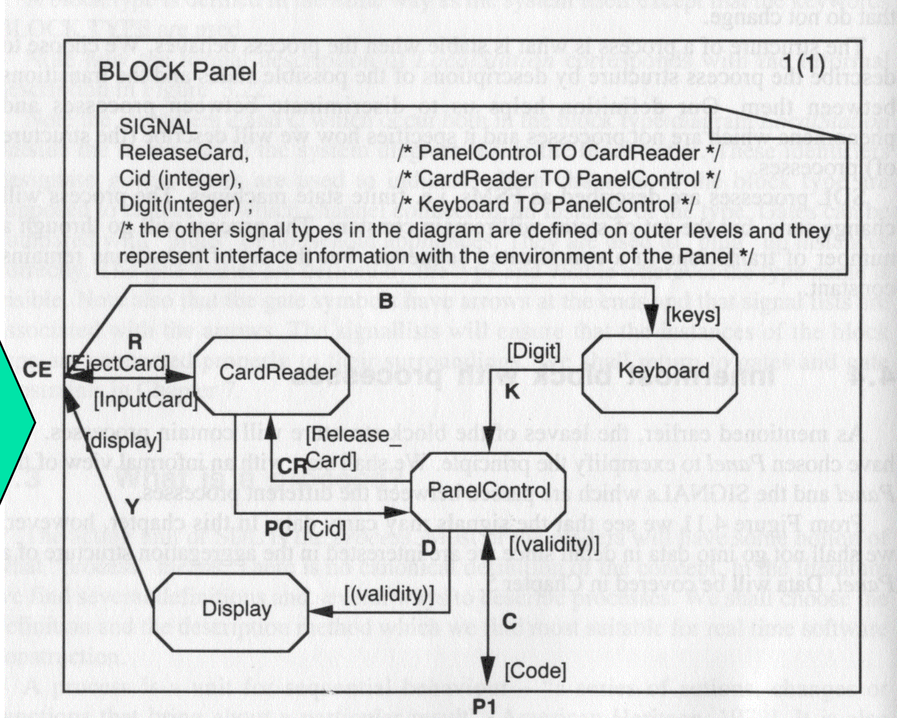
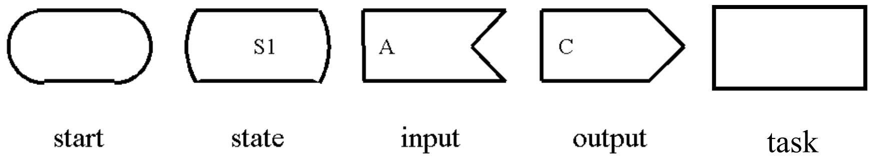
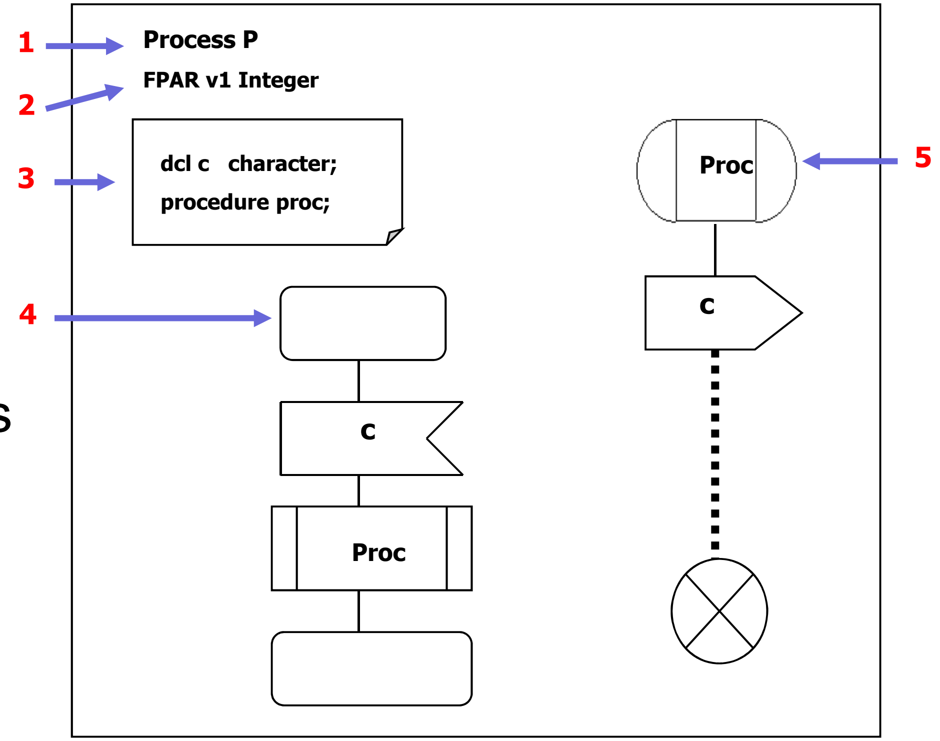


Figure 4.12 *Block Panel*

Process

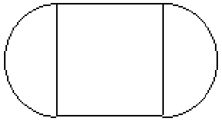
- Process in SDL is an extended FSM, in which the behavior of a FSM is described by states & transitions
- Process description is given through a process diagram
- In SDL, 5 basic constructs for the description of a process: start, state, input, output, and task
- Process diagram usually contains the following elements:
  1. Process name
  2. Formal parameters
  3. Variable descriptions
  4. Process graph
  5. Procedure Descriptions
  6. Timer descriptions (not in figure, see slide no. 29)



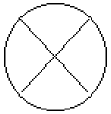


Procedure

- Procedure constructs similar to the programming languages
- Procedure is a FSM within a process. It is created when a procedure call is interpreted, and it dies when it terminates
- Procedure description is similar to a process description, with some exceptions
- Start symbol is replaced by the procedure start symbol



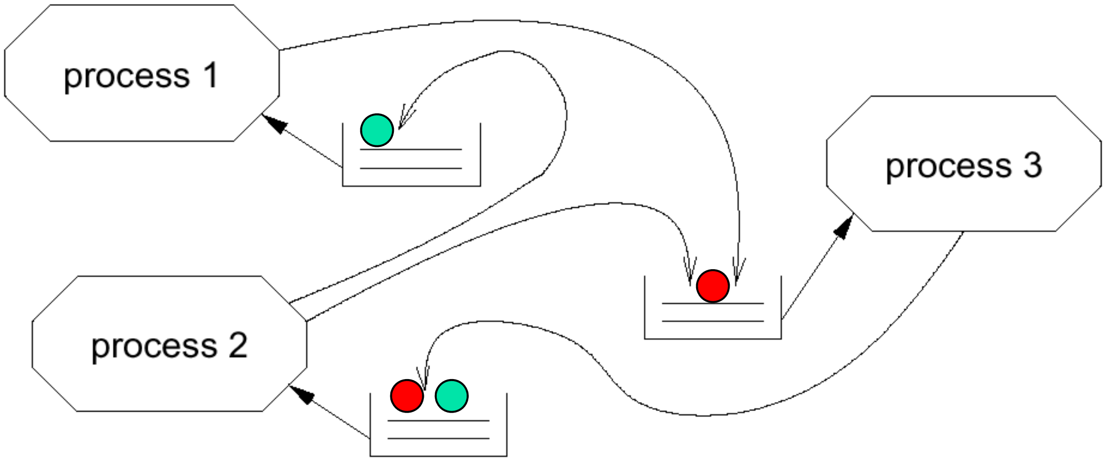
- Return symbol is introduced



- When a procedure is running, the calling process or procedure is suspended in the transition containing the procedure call

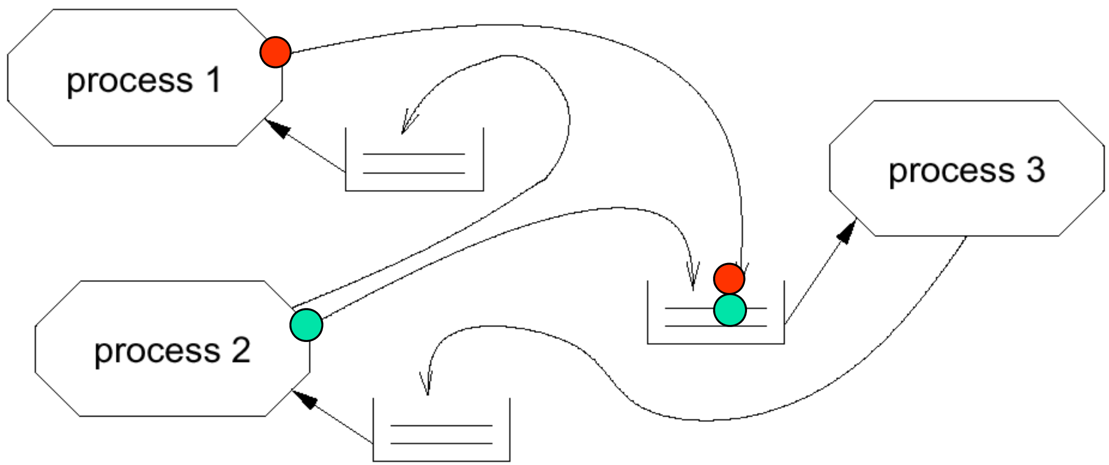
Communication in SDL

- Communication between FSMs (processes) is based on message-passing, assuming a potentially indefinitely large FIFO-queue
  - Each process fetches next entry from FIFO
  - Checks if input enables transition
    - If YES: transition takes place
    - If NO: input is ignored (exception: SAVE-mechanism)



Deterministic?

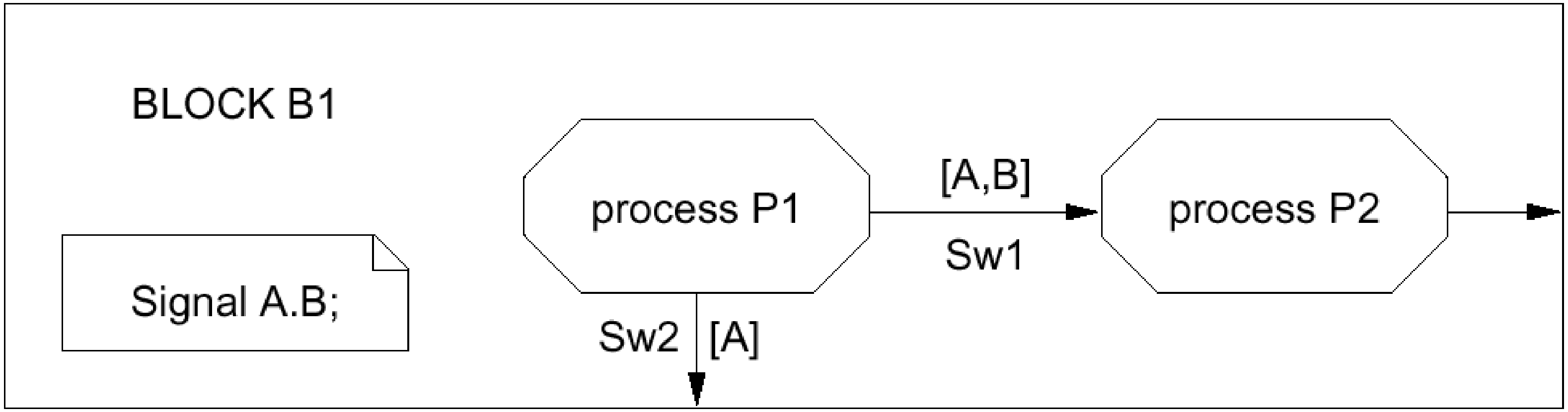
- Let tokens be arriving at FIFO at the same time:
  ☞Order in which they are stored, is unknown



All orders are legal: ☞simulators can show different behaviors for the same input, all of which are correct

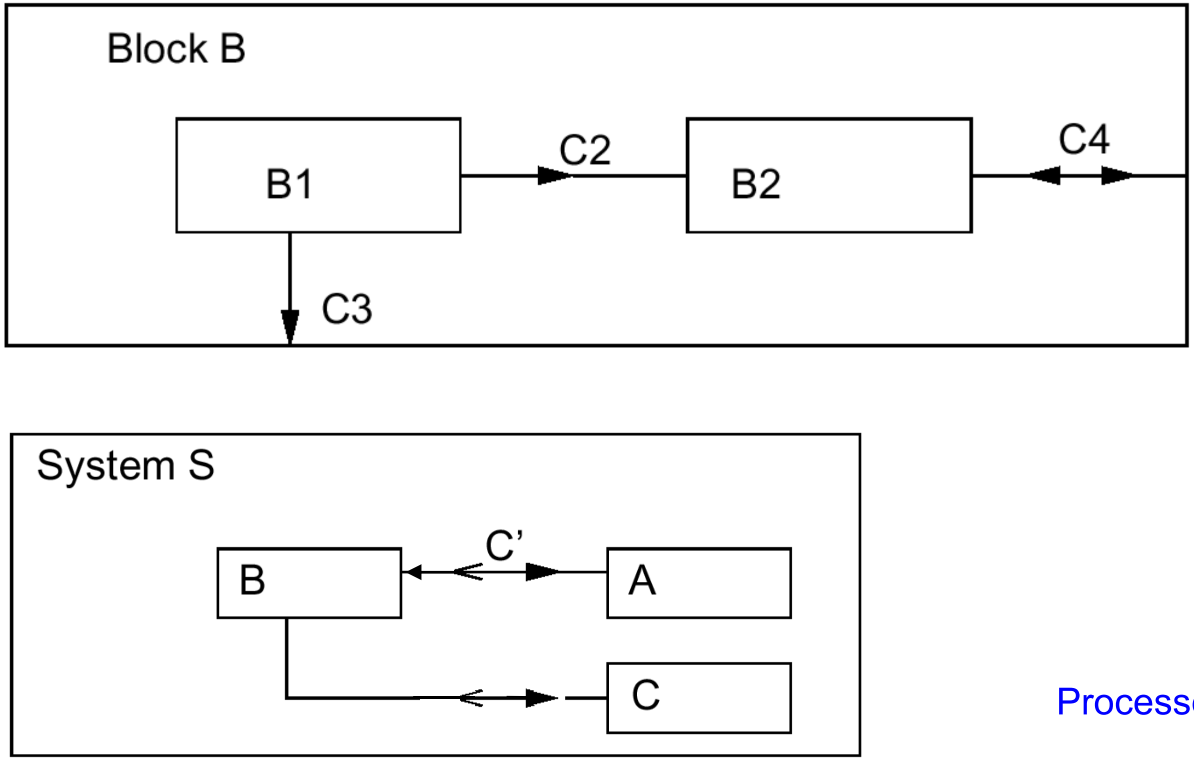
Process Interaction Diagram

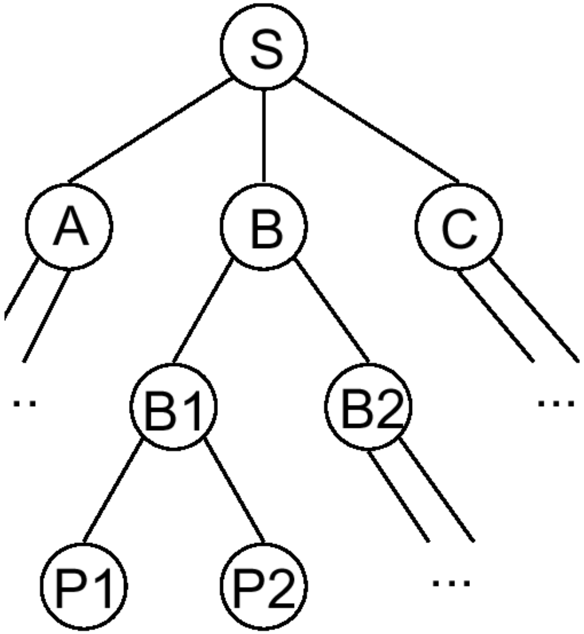
- Interaction between processes can be described in process interaction diagram
  - Special case of block diagrams
- Diagrams contain channels and declarations of local signals



Hierarchy in SDL

- Process interaction diagram can be included in blocks
- Root block is called system

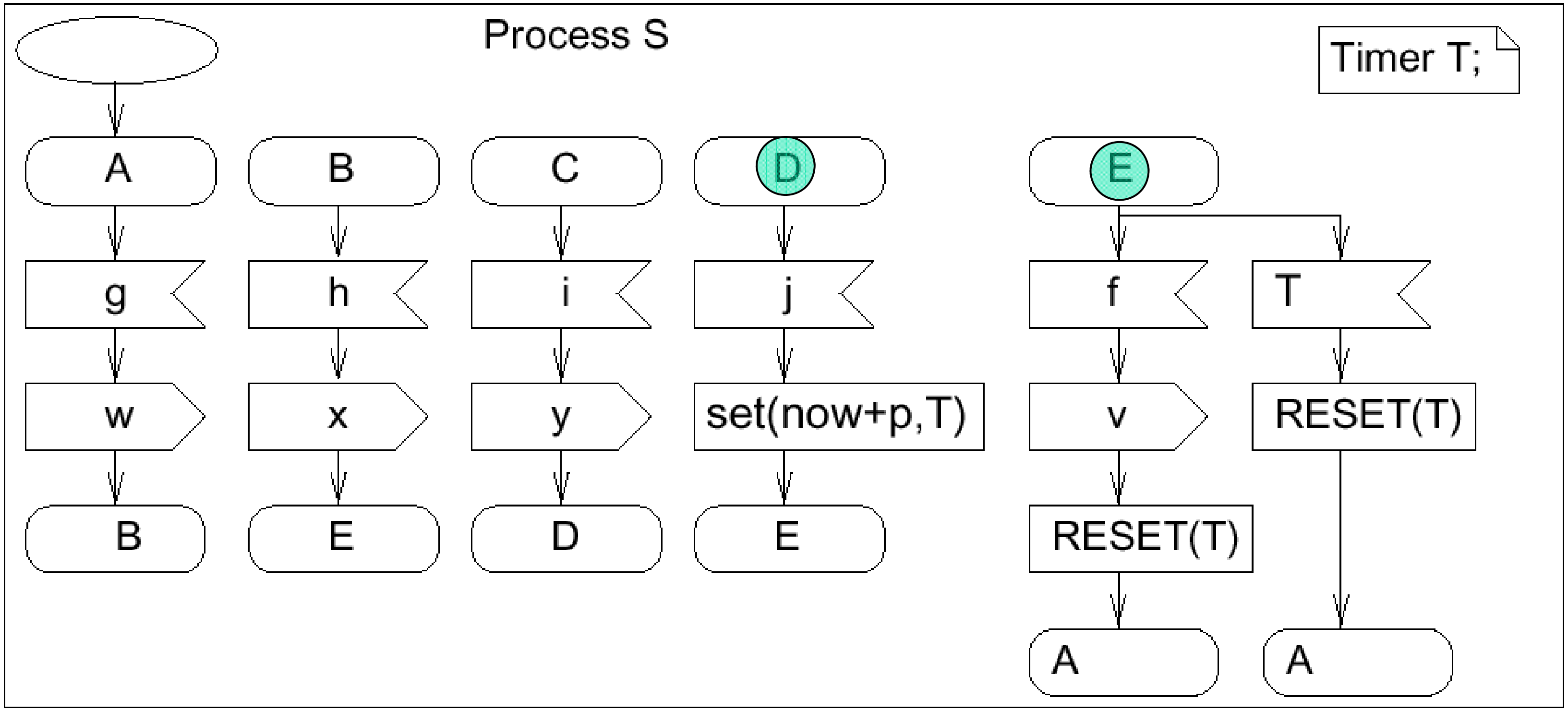




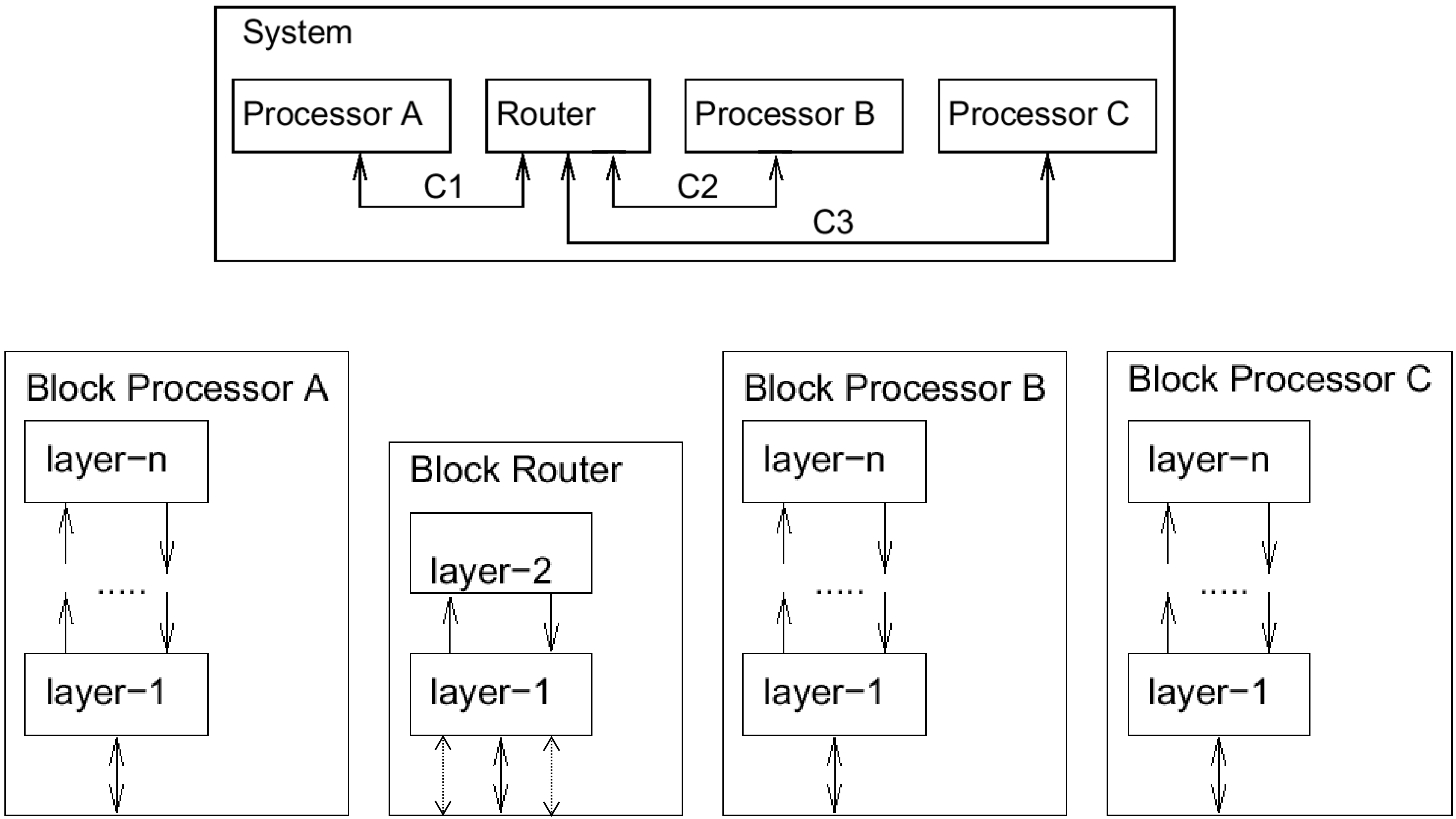
Processes cannot contain other processes, unlike in State Charts

Timer

- Timer can be declared locally. Elapsed timer puts signal into queue (not necessarily processed immediately)
- RESET removes timer (also from FIFO-queue)



SDL for Network Protocol Design

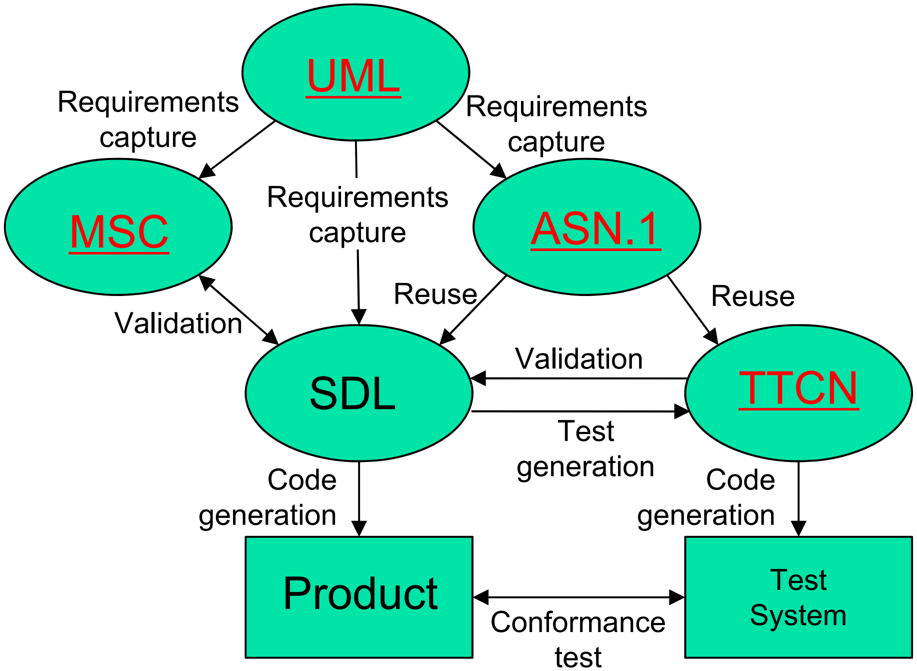


SDL: Summary

- Excellent for distributed applications
- Commercial tools available: SINTEF, Telelogic, Cinderella
- Not necessarily deterministic (Order in which FSMs are reading input is unknown)
  - No synchronous language
- Implementation requires bound for the maximum length of FIFOs; may be very difficult to compute
- Timer concept adequate for soft deadlines
- Disadvantages
  - Limited way of using hierarchies
  - Limited programming language support
  - No description of non-functional properties

SDL & Others SLs

- SDL is well-suited to be the core of full-scale projects because of its abilities to interface with other SLs



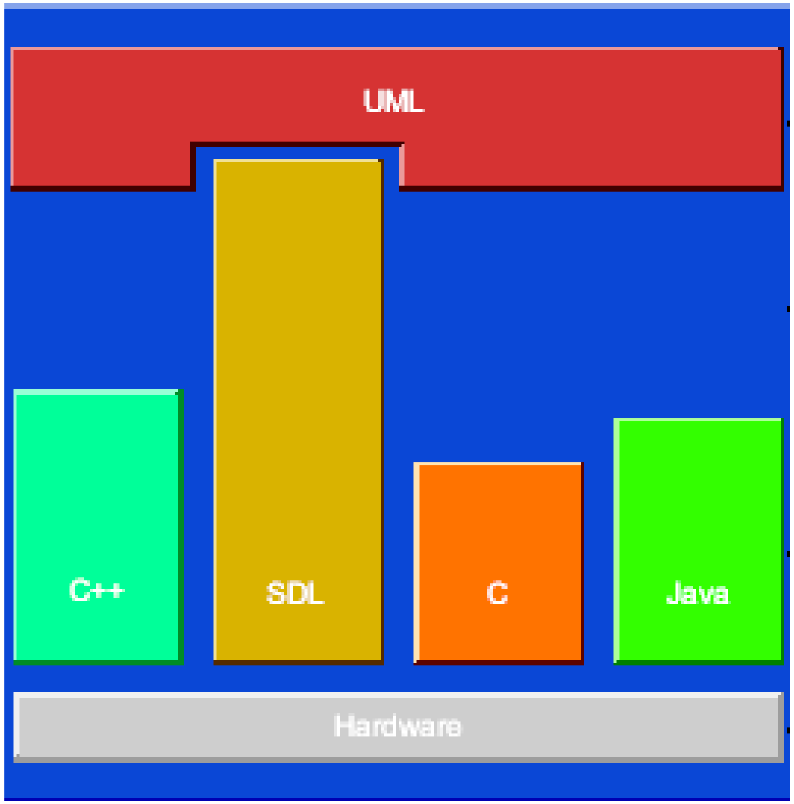
SDL & UML

- Unified modeling language (UML) is a language that is good at describing classes/types and relationships between them in a simple and intuitive manner
- It does a good job in the early phases of the development process but in the implementation phases it lacks in structural behavioral constructs
- SDL is more coherent than UML and also has a sound semantics foundation but is less expressive and widely accepted than UML
- However, SDL is strong exactly where UML is weak

Combining SDL with UML provides a modeling paradigm for visual software engineering that is more robust and effective than either language alone

SDL & UML (cont.)

- SDL bridges the gap between UML modeling and the target implementation



Requirements analysis

System specification

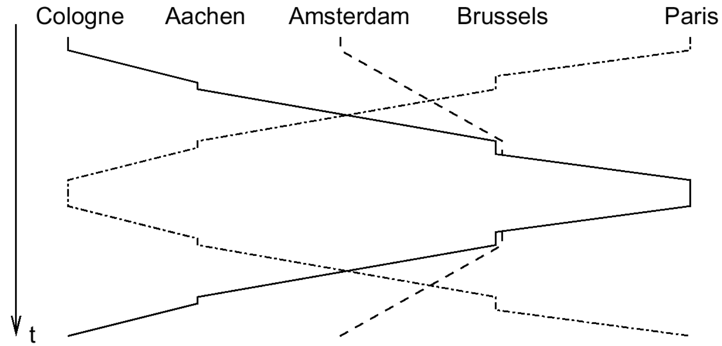
Design

Implementation

Target

MSC

- Message sequence chart (MSC) is a trace language for specification and description of the communication behavior of communicating systems
- Shows chronological sequences of messages sent between different system components (entities) and their environment
- Well-suited for presenting a complex dynamic behavior in a clear, unambiguous and easily understandable way
- Graphical means for representing schedules; time used vertically, geographical distribution horizontally
- No distinction between accidental overlap and synchronization



MSC for Connecting Sensor

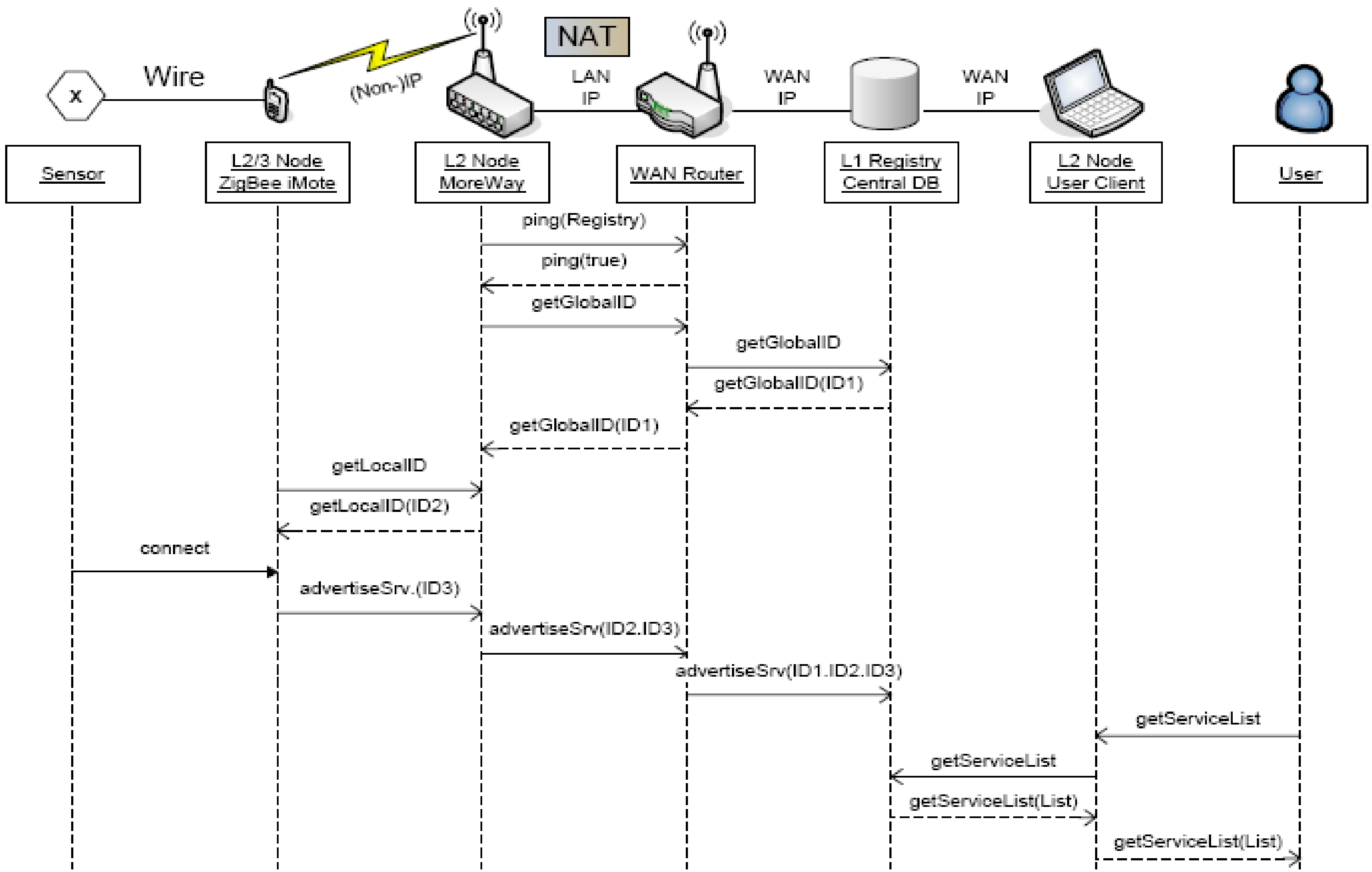


Fig. Message dialogue

MSC Discussion

- Pros:
  - Appropriate for visualizing schedules
  - Proven method for representing schedules in transportation
  - Standard defined: *ITU-TS Recommendation Z.120:* ITU-TS, Geneva, 1996

- Cons:
  - Describes just one case, no timing tolerances: *“What does an MSC specification mean: does it describe all behaviors of a system, or does it describe a set of sample behaviors of a system?”*

ASN.1

- Abstract syntax notation one (ASN.1) is the ISO-ITU standardized language for defining data types and values
- Designed for describing **structured information** that is conveyed across some interface or communication medium, independently from the transfer format
- Used in conjunction with Basic Encoding Rules (BER), defining the so-called "transfer syntax", the representation of values as a bit stream to be transmitted
- Features
  - Formal
  - Standardized
  - Complete data type definition apparatus
  - Constraints, class definitions, and parameterization
  - Extensibility
  - Automatic encoder/decoder generation
  - Automatic implementation language representation generation

ASN.1 Example

```
PersonName ::= [16] IMPLICIT OCTET STRING

Person ::= SEQUENCE {
     title  [1] OCTET STRING,
     name   [2] PersonName,
     height [3] Height OPTIONAL,
     weight [4] INTEGER OPTIONAL
}

AddressBookEntry ::= CHOICE {
     [10] Person,
     [11] Company,
     [12] Group
}
```

TTCN

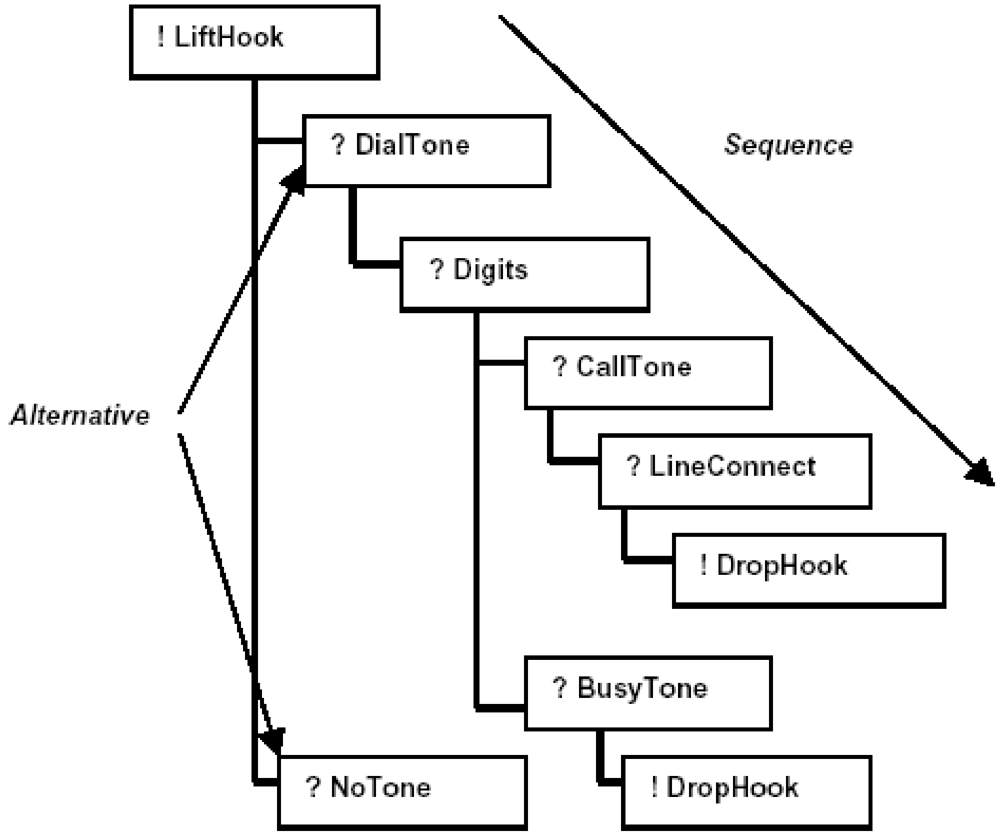
- Tree and tabular combined notation (TTCN) is a globally adopted standard test notation for the specification of test cases
- In TTCN, a test case is used for testing special functionality such as non-termination
- Syntax is similar to conventional programming languages and incorporates features like synchronous and asynchronous communication mechanisms as well as dynamic parallel test configuration
- TTCN test suite consists of four parts:
  - Overview part: An index and page references of the different part
  - Declaration part: Declaration of PDUs with ASN.1
  - Constraints part: Describe the value that re sent or received
  - Dynamic part: Describe the actual execution behavior of test suite and contains all test cases and default table for

TTCN Features

- Portability
  - TTCN is platform independent
- Reusability
  - Since an abstract test suite uses only the external (standardized) interfaces, it is highly reusable
- Modularity
  - Test cases can share a common set of definitions, and be added incrementally
- ASN.1 integration
  - ASN.1 is used for representation of data (PDUs)

TTCN Example 1

- Each test case (or test step) is called behavior tree



TTCN Example 2

- A behavior table describes the behavior tree in a tabular format

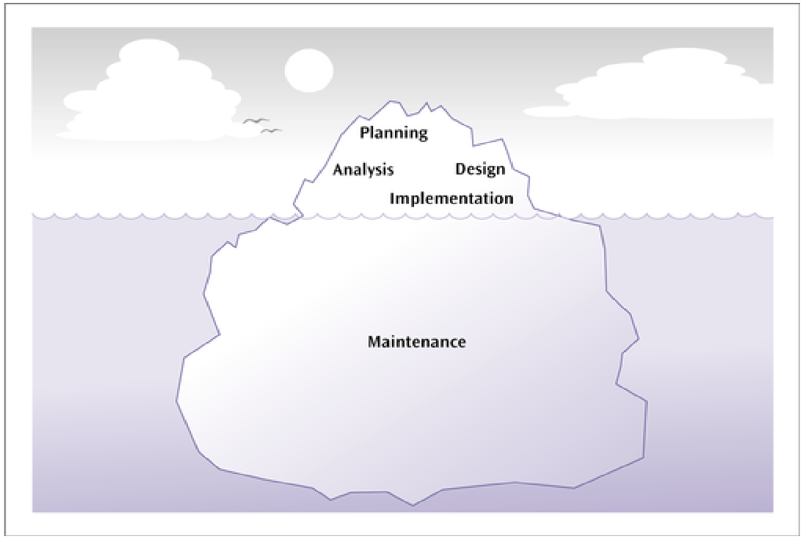
Test Case Dynamic Behavior				
Test Case Name: Basic Connect				
Purpose: Check that a normal Call connection can be established				
Nr	Label	Behavior Description	Cref	V
1		! LiftHook		
2		? DialTone		
3		! Digits	CallSubsr2	
4		? CallTone		
5		? LineConnect	ConnSubscr2	
6		! DropHook		P
7		? BusyTone		
8		! DropHook		I
9		? NoTone		F
Detailed Comments:				

Verdict (decision): *Pass (P), Inconclusive (I), Fail (F)*

Computer Network Design

- Properly designing a computer network is a difficult task. It requires planning and analysis, feasibility studies, capacity planning, and baseline creation skills

- Performing network management is difficult too. A network manager must possess computer and people skills, management skills, financial skills, and be able to keep up with changing technology



- Network design
  - Determines the location and type of network devices, the types and size of communication links to provide services to the electronic communication devices

Components of a Network

- Server
  - Computer that provides services to other networked computers
- Client
  - Computer in a client/server relationship, e.g., web browser
- Hardware/Software
  - Network card, router, modem, hub/NT operating systems, utilities
- Media
  - The way to connect computers on a network
- Data
  - Files to be shared by network computers
- Resources
  - Peripherals, e.g., printers, to be used by network computers

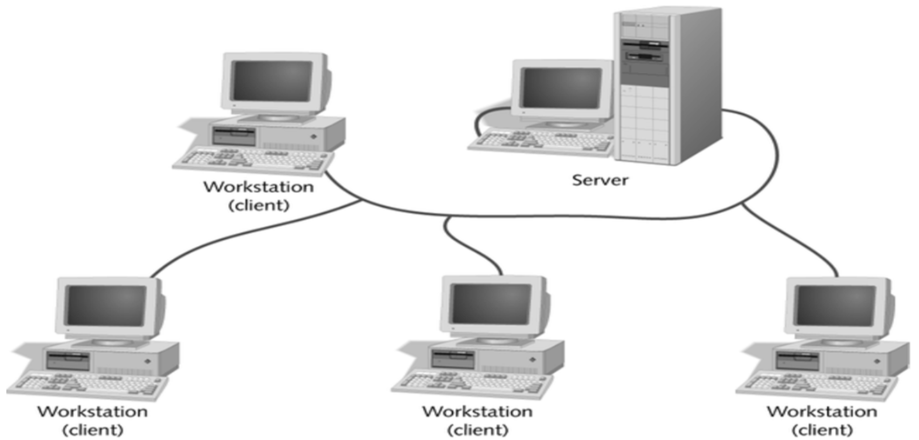
Media and Hardware of a Network

- Media:
  - Coaxial cable
  - Twisted pair cable
  - Fiber optic cable
  - Microwave
  - Communications satellite
  - Cellular phones
- Hardware:
  - Workstations
  - Servers
  - Bridges
  - Routers
  - Hubs and switches
  - Terminals

Network Models

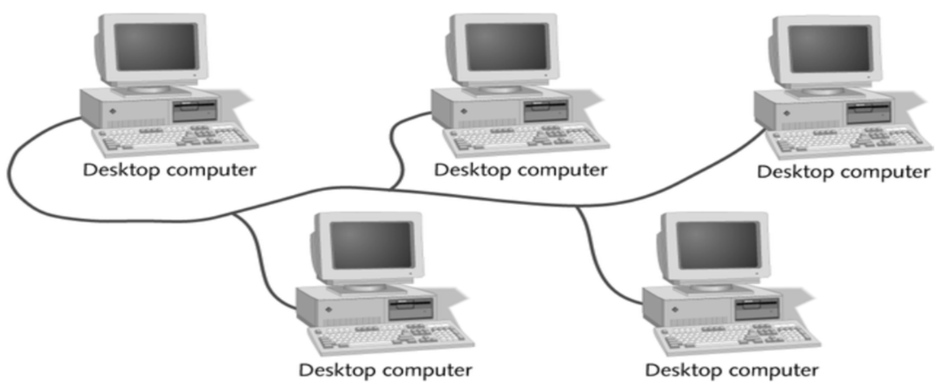
Client/Server Model

Microcomputer users, or *clients*, share services of a centralized computer called a *server*



Peer-to-Peer Model

Computers share equally with one another without having to rely on a central server



Client/Server vs. Peer-to-Peer

Client/Server Model

Advantages:

- Very secure OS
- Better performance
- Centralized servers, easy to manage
- Centralized backups
- High reliability

Disadvantages:

- Expensive administration
- More hardware intensive

Peer-to-Peer Model

Advantages:

- Uses less expensive networks
- Easy to administer
- Contain both network operating system and application software
- Ideal for small business and home users

Disadvantages:

- Individual user performance easily affected
- Not very secure
- Hard to back up

Network Design Selection Criteria

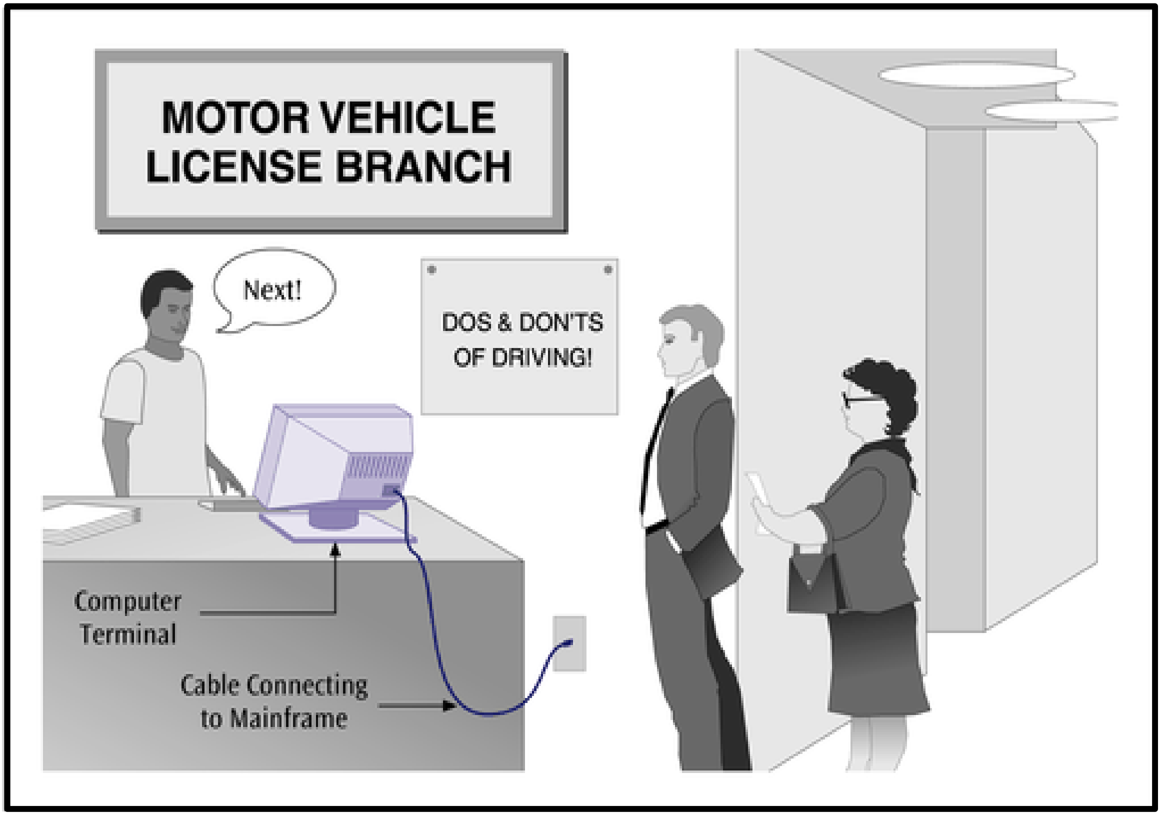
- Size of the organization
- Level of required security
- Level of available administrative support
- Amount of network traffic
- Needs of the network users
- Budget for building the network

Computer Network Configurations

- Connections between networks
  - Computer terminal to mainframe computer
  - LAN to LAN connection
  - Sensor to LAN connection
  - Wireless telephone connection
  - Microcomputer to mainframe computer
  - Microcomputer to LAN
  - Microcomputer to Internet
  - LAN to WAN connection
  - Satellite and microwave

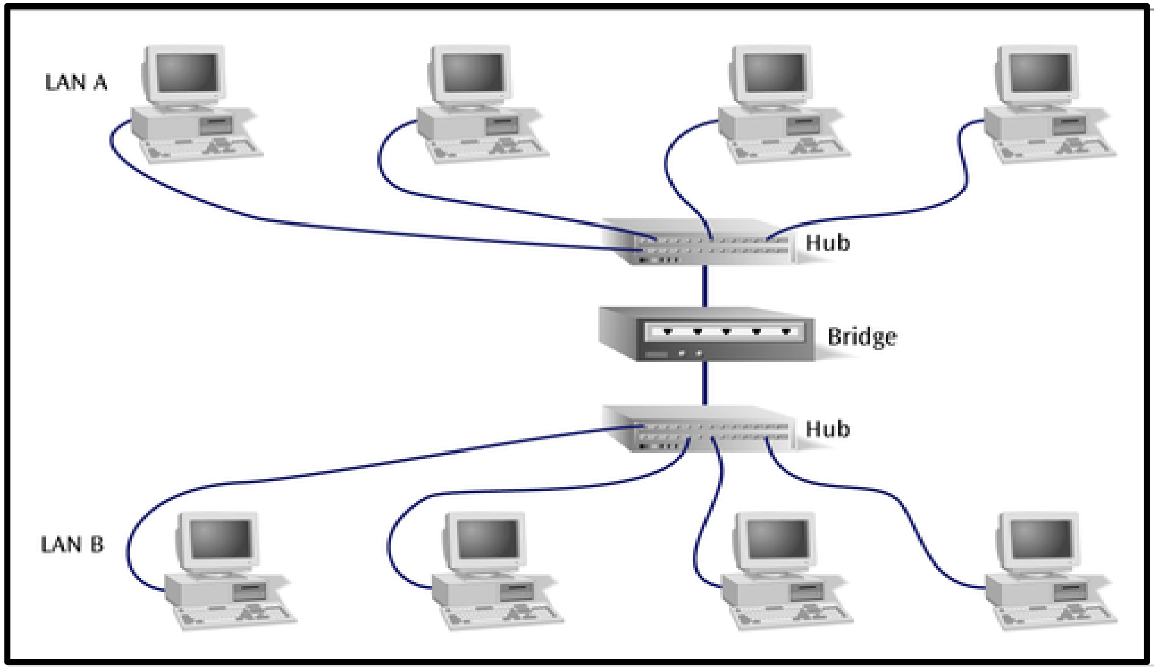
Computer Terminal to Mainframe Computer

- Used in many types of businesses for data entry and data retrieval
- Usually involves a low-speed connection



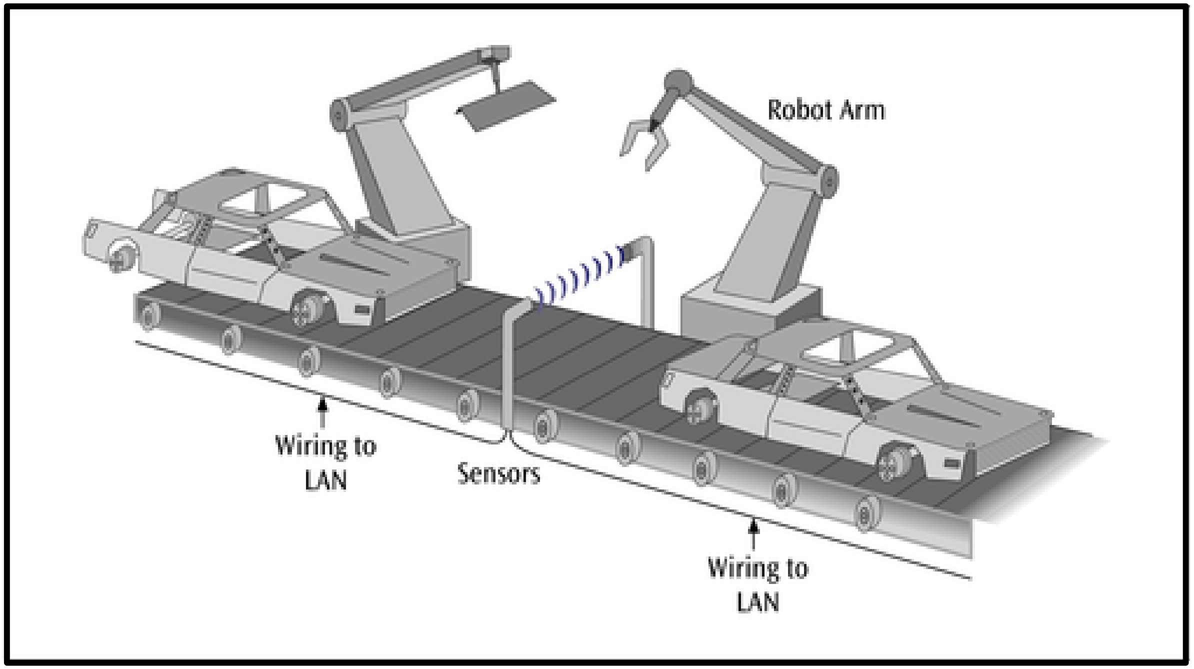
LAN to LAN Connection

- Found in businesses and schools that have two or more LANs and a need for them to intercommunicate
- Bridge is a typical device used to interconnect LANs



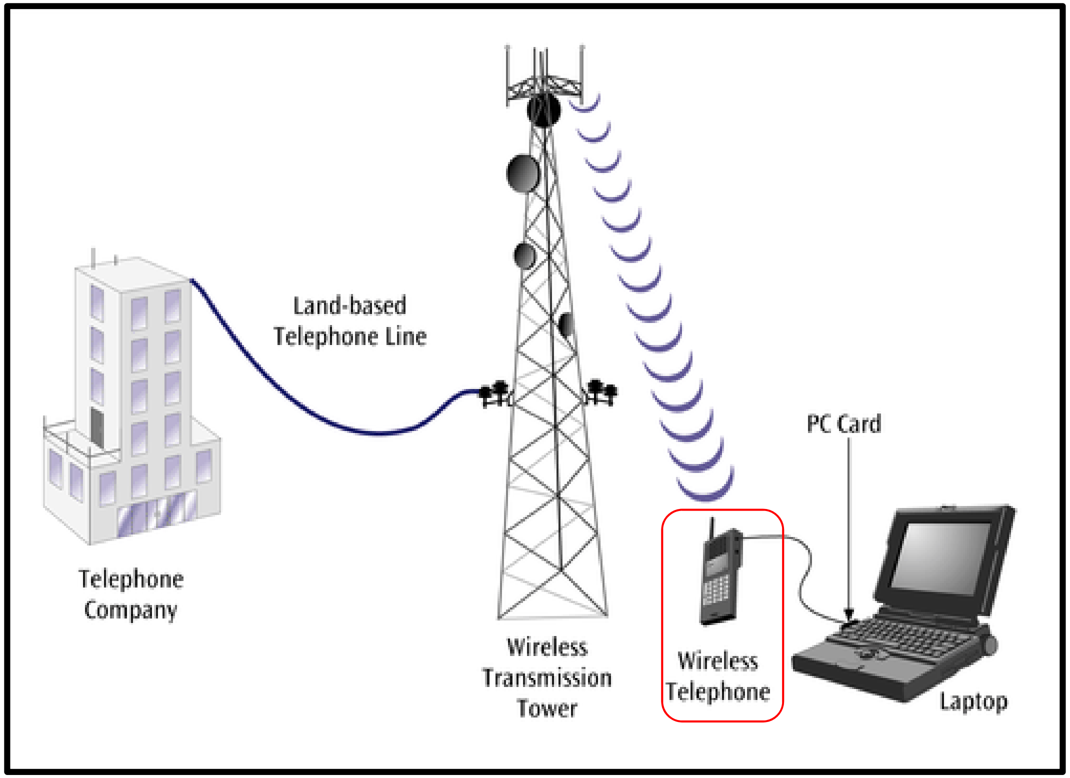
Sensor to LAN Connection

- Found in industrial environments
- Assembly lines and robotic controls depend heavily on sensor-based local area networks



Wireless Telephone Connection

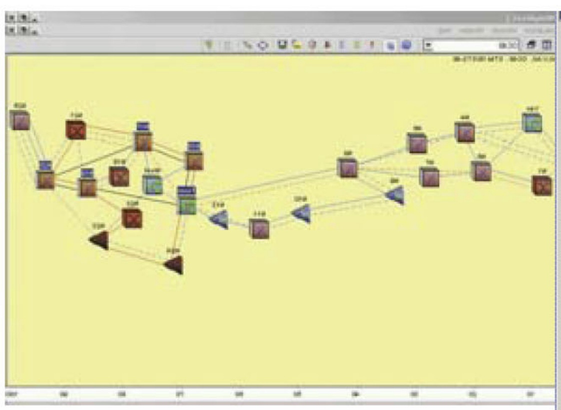
- Quickly expanding market across worldwide
- 1G: First generation analog services
- 2G: Second generation PCS services available in most areas and under many types of plans
- 3G: Third generation services beginning to appear in Europe and Asia

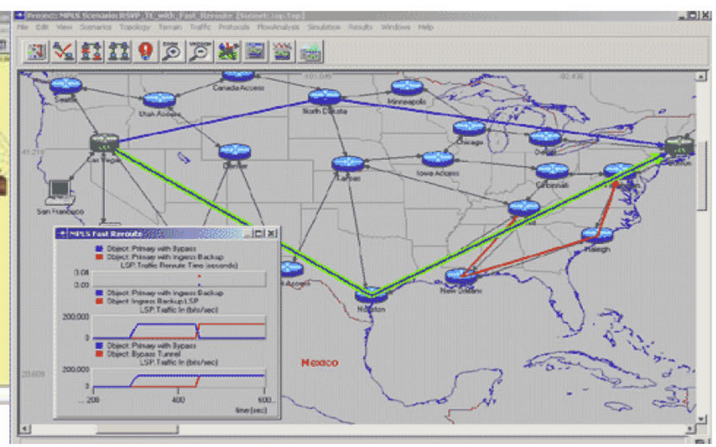


PCS = personal communications service

Network Design Tools

- Computer aided design tools available for certain problems
  - Aimed at the network design of Metropolitan, Backbone and Wireless Access
  - User provides set of traffic demands, geographic locations, performance requirements, etc
  - Most use some sort of optimization based formulation with heuristic solution to minimize cost
- Variety of tools available – many are internal vendor or consultant tools:
  - OnePlan by VPISystems (wired networks)
  - SP-Guru by OPNET (wired networks)
  - ETX by ETX (cellular networks)
  - CellPlan by CellCAD (cellular networks)





Announcement

- Next is Chapter 12 Design, Implementation, and Operation of Network Systems
- 09:00 ~ 10:40 on 21 November (Monday)