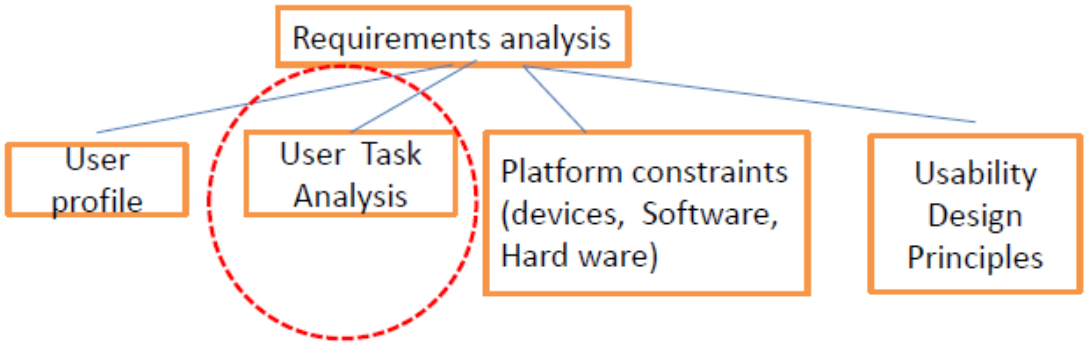
UNIT - V
Task modeling and analysis

Basics of Hierarchical Task Analysis (HTA)

- Task Analysis forms an important part of User Requirements Analysis.



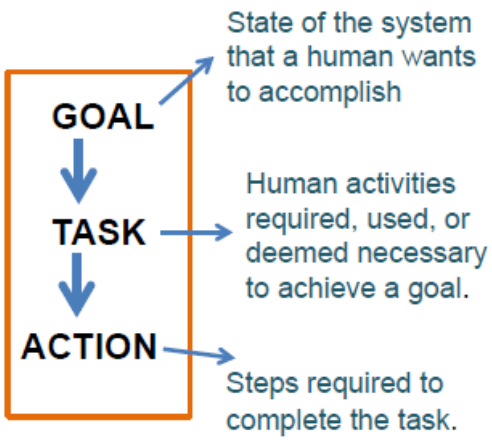
- Task analysis is a study of users, work flow patterns, conceptual frame works, & sequential execution of interaction with the GUI.
- Task analysis results in an user's mental map of how he / she breaks down 'goals' into a series of smaller tasks & sequences them.

Task analysis focuses on understanding 'User'

- Users' goals and how they achieve them.
- Personal, social, and cultural characteristics, users bring to their tasks.
- Physical environment's influence on users.
- The influence of previous knowledge and experience on: How users think about their work.
- The workflow users follow to perform their tasks.

What is a 'TASK'?

- A set of human actions that contributes to a functional objective and to the goal of the system.
- Task analysis defines performance of humans & not computers.



- Task Analysis means understanding User's needs, sequencing them into a series of hierarchical acts (interactions) so as to facilitate the achieving of the goal.

Illustration: Making a phone call

User's need : To communicate with a particular individual
Goal is to inform & seek feed back from that individual in a particular context.
Actions: Putting 'ON' the Phone; dialing the number; communicating; ending the call.

- There is a sequence and a hierarchy of actions to be followed. One cannot go to the next step in making calls unless the previous one is complete.

Task Analysis includes:

- User's goal; user's need; user's intentions.
- Understanding user's environment –context of use.
- Planning for the 'actions'

- Task analysis has direct implications in software design
- Hierarchical Task Analysis is decomposing tasks into subtasks & analyzing the logic of sequence needed to execute the task to achieve the set goal (state) in an optimal way.

"A **hierarchy** is an organization of elements that, according to prerequisite relationships, describes the path of experiences a learner must take to achieve any single behavior that appears higher in the hierarchy . (Seels & Glasgow, 1990, p. 94)".

Techniques for analysis

- Task decomposition (split tasks into subtasks in sequence)
- Knowledge-based techniques (what users need to know)

Involves description of tasks in terms of

- Goals (or states) they achieve after execution
- Steps involved
- Relevant contextual information

Task decomposition example
(Split tasks into subtasks in sequence)

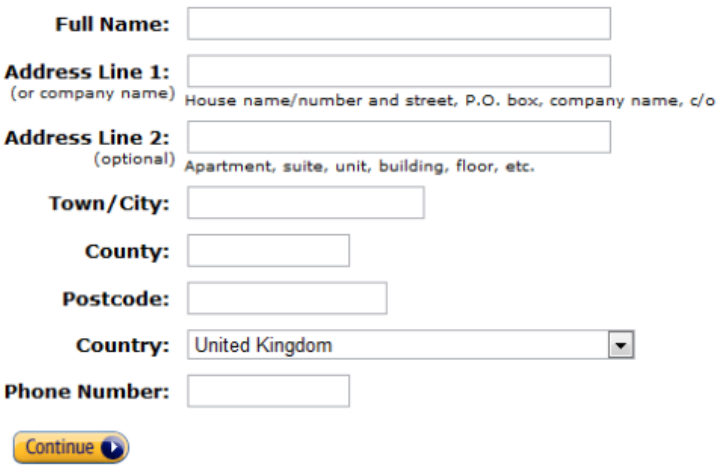
HTA provides a consistent logical description of the interdependencies of tasks and therefore forms a rational framework for description of possible user interface architecture based on which a GUI is visualized.

Task 1: Feed in Address information to order book

Locate the **Full Name** field.
Move the insertion point to the field.

Sub task 1.1

Type the full name.
Action 1
Locate the **Address Line 1** field.
Action 2
Move the insertion point to the field.
Action 3
Type the address.
Optional: Locate the **Address Line 2** field.
Action 4
Move the insertion point to the field.
Action 5
Type the address.
Sub Task 1.2
Locate the **Town/City** field.
Move the insertion point to the field.
Type the Town or City
Sub Task 1.3
Locate the County field.
Move the insertion point to the field.
Type the county.
Sub Task 1.4
Locate the Postcode field.
Move the insertion point to the field.
Type the postal code.
Sub Task 1.5
Locate the Country field.
Move the insertion point to the field.
Select the country from the drop-down list.
Sub Task 1.6
Locate the Phone Number field.
Move the insertion point to the field.
Type the phone number.



The diagram shows how the high level steps of a task relate to one another. The structured breakdown of the task into its subtasks describes each interaction in detail.

Collecting user data for Task Analysis

- **Ethnography:** Observing and noting users behavior in the use context
- **Protocol analysis:** Observing and documenting actions of the user by validating user's mental thinking.

  Methods for recording user actions include the following:

  Paper and pencil: This is primitive, but cheap, and allows the analyst to note interpretations and extraneous events as they occur. However, it is hard to get detailed information, as it is limited by the analyst's writing speed. A variation of paper and pencil is the use of a notebook computer for direct entry, but then one is limited to the analyst's typing speed, and one loses the flexibility of paper for writing styles, quick diagrams and spatial layout. If this is the only recording facility available then a specific note-taker, separate from the evaluator, is recommended.

  Audio recording: This is useful if the user is actively 'thinking aloud'. However, it may be difficult to record sufficient information to identify exact actions in later analysis, and it can be difficult to match an audio recording to some other form of protocol (such as a handwritten script).

  Video recording: This has the advantage that we can see *what* the participant is doing (*as long as* the participant stays within the range of the camera). Choosing suitable camera positions and viewing angles so that you get sufficient detail and yet keep the participant in view is difficult. Alternatively, one has to ask the participant not to move, which may not be appropriate for studying normal behavior! For single-user computer-based tasks, one typically uses two video cameras, one looking at the computer screen and one with a wider focus including the user's face and hands.

  Computer logging: It is relatively easy to get a system automatically to record user actions at a keystroke level, particularly if this facility has been considered early in the design. It can be more difficult with proprietary software where source code is not available (although some software now provides built-in logging and playback facilities). Obviously, computer logging only tells us what the user is doing on the system, but this may be sufficient for some purposes. Keystroke data are also 'semantics free' in that they only tell us about the lowest-level actions, not why they were performed or how they are structured (although slight pauses and gaps can give clues).

  Automatic Protocol Analysis Tool: *EVA* (*Experimental Video Annotator*) is a system that runs on a multimedia workstation with a direct link to a video recorder. The evaluator can devise a set of buttons indicating different events. These may include timestamps and snapshots, as well as notes of expected events and errors. The buttons are used within a recording session by the evaluator to annotate the video with notes. During the session the user works at a workstation and is recorded, using video and perhaps audio and system logging as well.
- **Think aloud and cooperative evaluation**

  Think aloud is a form of observation where the user is asked to talk through what he is doing as he is being observed; for example, describing what he believes is happening, why he takes an action, what he is trying to do.

  Think aloud has the advantage of simplicity; it requires little expertise to perform (though can be tricky to analyze fully) and can provide useful insight into problems with an interface. It can also be employed to observe how the system is actually used.

  It can be used for evaluation throughout the design process, using paper or simulated mock-ups for the earlier stages. However, the information provided is often subjective and may be selective, depending on the tasks provided. The process of observation can alter the way that people perform tasks and so provide a biased view.

- **Interviews**

  Interviewing users about their experience with an interactive system provides a direct and structured way of gathering information. Interviews have the advantages that the level of questioning can be varied to suit the context and that the evaluator can probe the user more deeply on interesting issues as they arise. An interview will usually follow a top-down approach, starting with a general question about a task and progressing to more leading questions (often of the form 'why?' or 'what if ?') to elaborate aspects of the user's response.

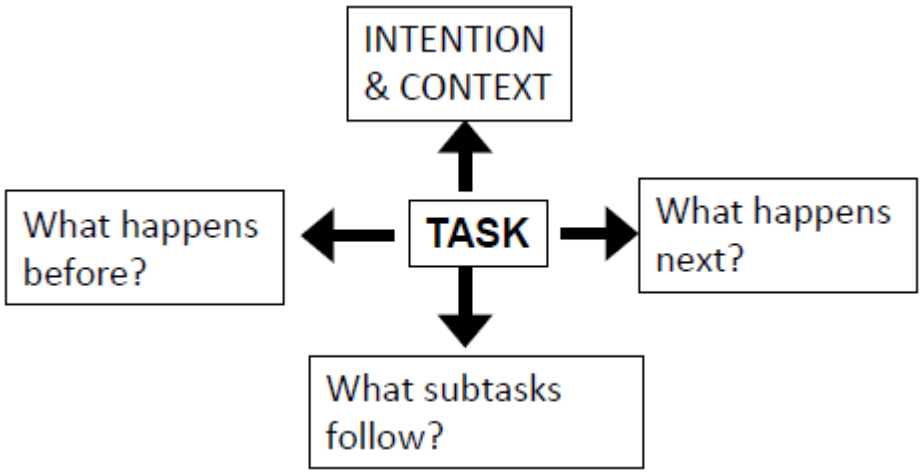
  In order to be as effective as possible, the interview should be planned in advance, with a set of central questions prepared. Each interview is then structured around these questions. This helps to focus the purpose of the interview, which may, for instance, be to probe a particular aspect of the interaction. It also helps to ensure a base of consistency between the interviews of different users.

- **Questionnaires**

  An alternative method of querying the user is to administer a questionnaire. This is clearly less flexible than the interview technique, since questions are fixed in advance, and it is likely that the questions will be less probing. However, it can be used to reach a wider participant group, it takes less time to administer, and it can be analyzed more rigorously. It can also be administered at various points in the design process, including during requirements capture, task analysis and evaluation, in order to get information on the user's needs, preferences and experience.

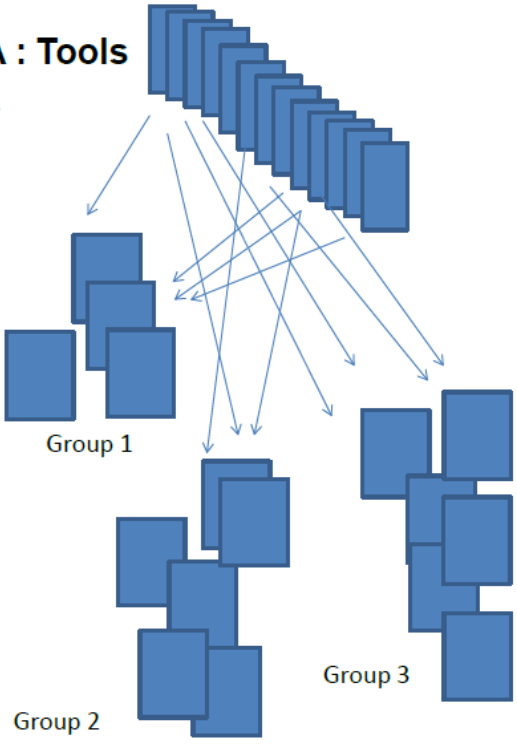
  Given that the evaluator is not likely to be directly involved in the completion of the questionnaire, it is vital that it is well designed. The first thing that the evaluator must establish is the purpose of the questionnaire: what information is sought? It is also useful to decide at this stage how the questionnaire responses are to be analyzed.

  There are a number of styles of question that can be included in the questionnaire. These include the following: **General, Open ended, Scalar, Multi choice, Ranked.**



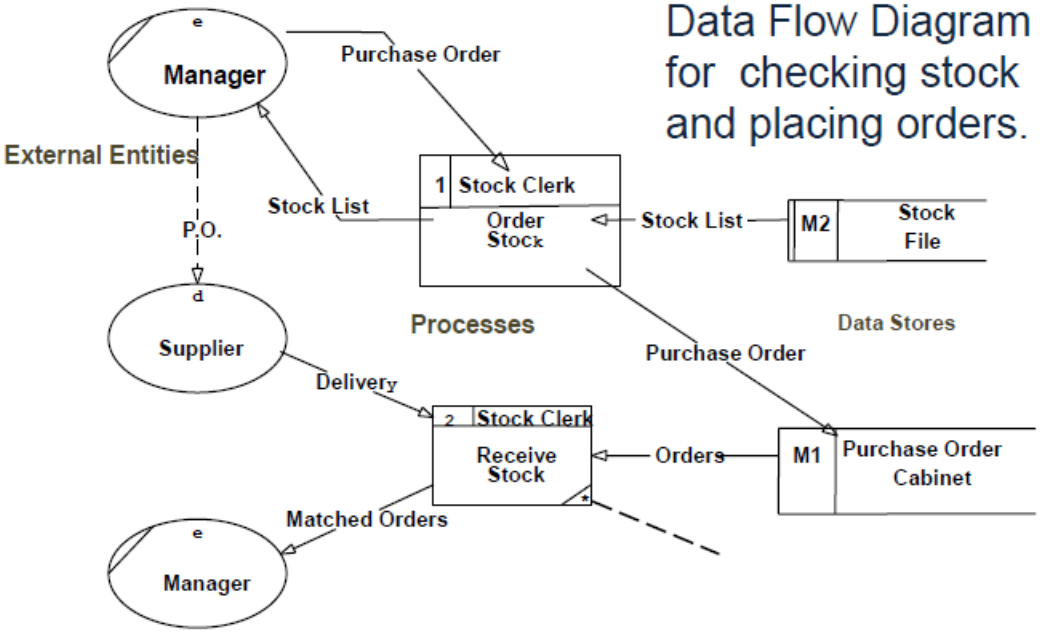
Modeling user data for HTA : Tools

- **Affinity Diagrams:** Similar data or similar actions are grouped together into categories till a pattern emerges in the form of a diagram.
- **Stick notes or cards** are used to scribble labels. These are grouped and regrouped till a pattern that shows affinity of different groupings becomes evident.
- The degree of affinity is used while determining hierarchy of actions or hierarchy of information.



Modeling user data for HTA: Tools

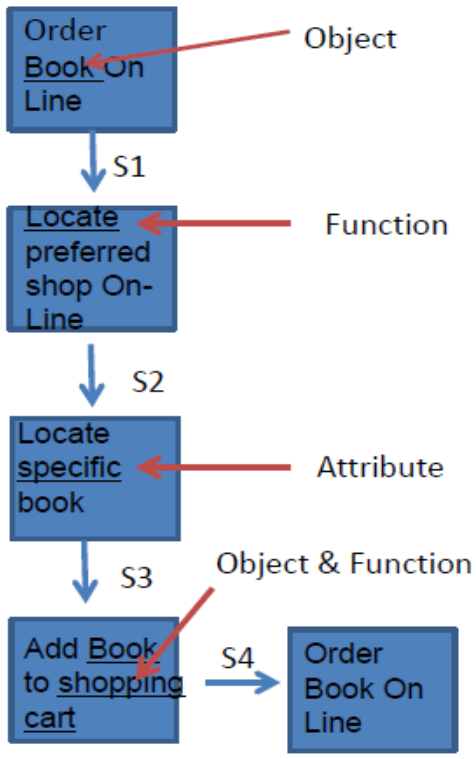
Flow Diagrams: Indicate flow of information through a system. They illustrate dependency of system elements (states) and how information moves -one from another.
They can also be indicative of roles that are assigned within an organization and how data moves between these assigned roles as well as between organization as a whole & the outside world.
An example of a Flow Diagram showing flow of information in an organization executing the task of checking stock and ordering supplies is shown in the next slide.
The diagram is called as DFD-short for Data Flow Diagram and is a standard form of depiction used in Information Systems Design in Systems Engineering.



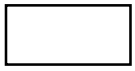
Data Flow in the above (part) diagram depicts a Manager calling for Stock list from the stock clerk who gets it from the database M2. Purchase order is placed and filed in Database M1. Clerk 2 receives stock from supplier and acts further (dotted line).

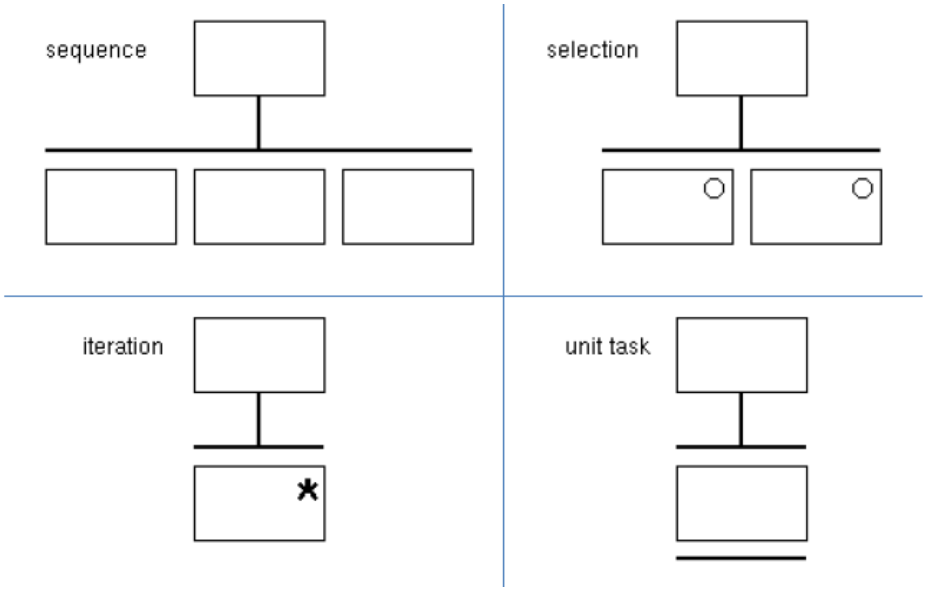
Sequence Diagrams:

Sequence diagrams are procedural analysis diagrams. While flow diagrams track work through a system, a sequence diagram uses TIME to track actions & decisions. Sequence diagrams are critical because they give the OBJECTS, FUNCTIONS & ATTRIBUTES of a system which in turn are used to derive the UI information Architecture.

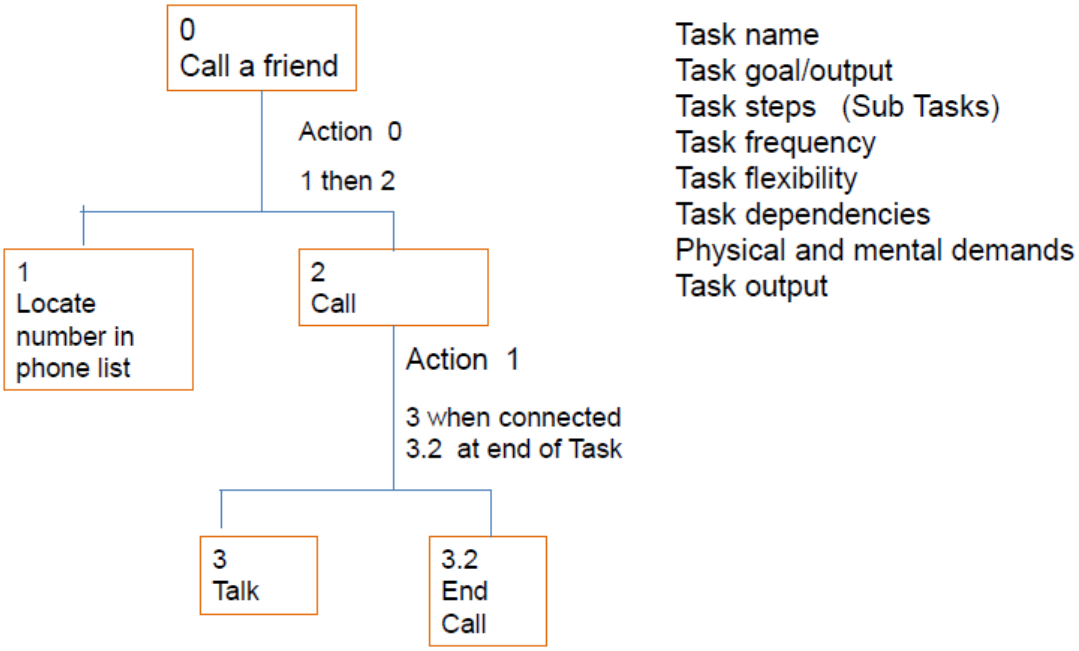


HTA Structure Chart Notation

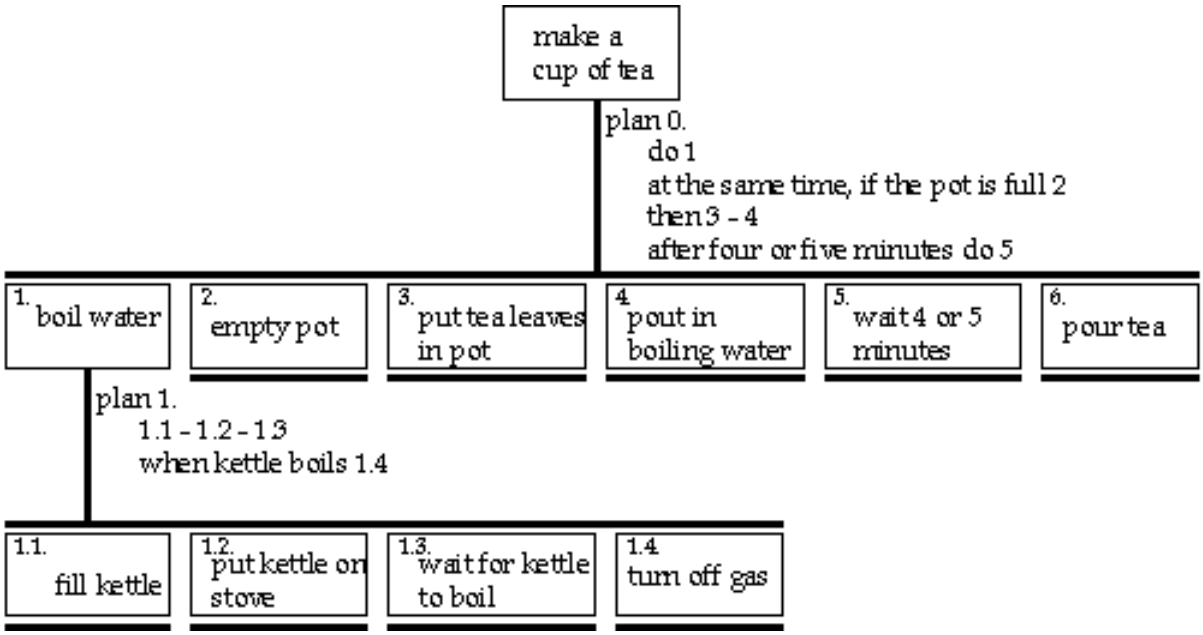
- Activities 
- Flow of control (Downward)
- Sequencing (Left to Right)
- Repetition (*)
- Selection (0)



Example 1: HTA Diagram for making a Phone Call



Example 2: HTA Diagram for Tea-making (Structure chart notation)



Tea-making HTA model in (Textual notation)

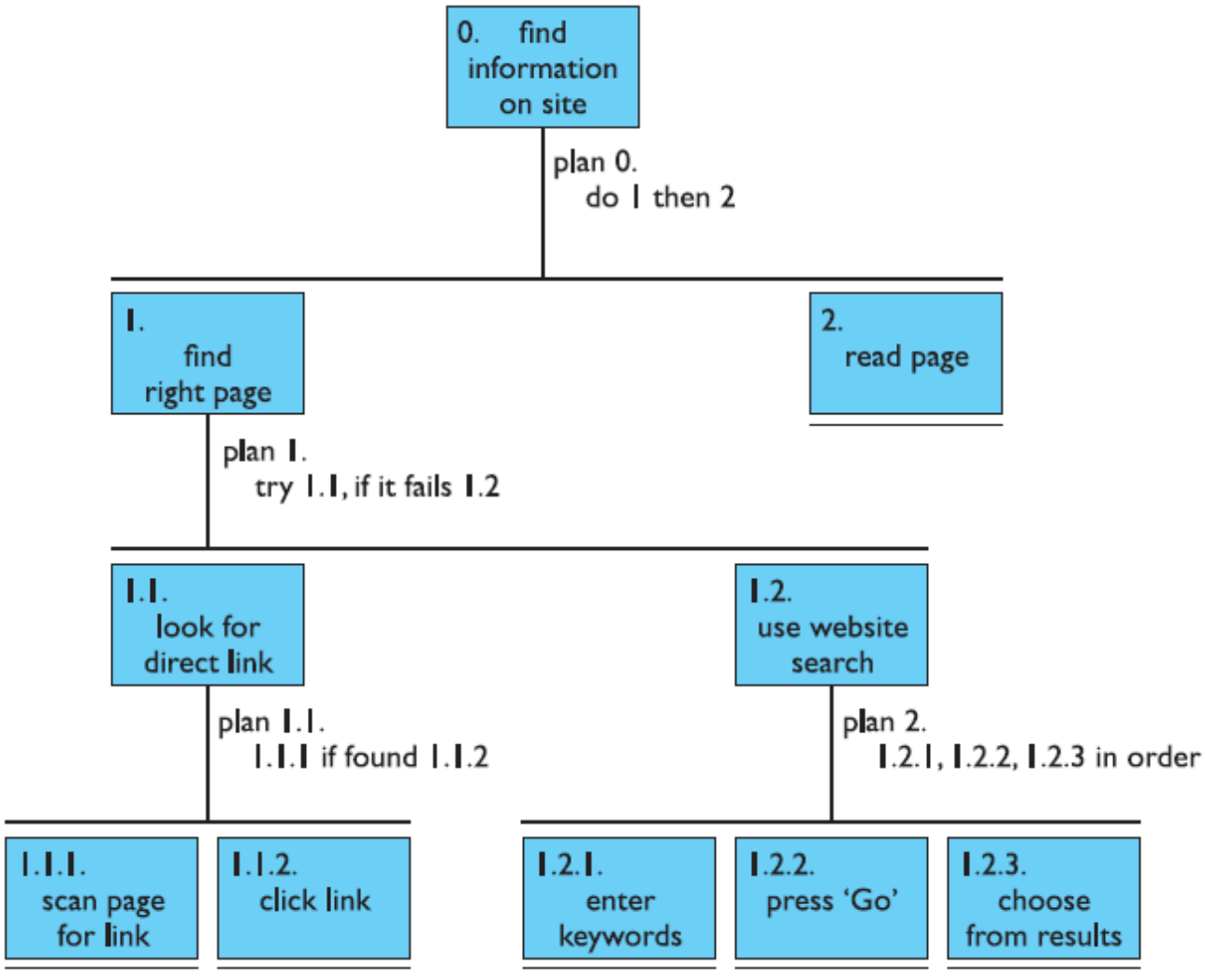
0. Make tea
   1. Boil water
      1.1 fill kettle
      1.2 light stove
      1.3 put kettle on stove
      1.4 wait
      1.5 turn off stove

2. Empty pot
3. Put leaves in pot
4. Pour water
5. Wait
6. Pour tea

Plan 0: do 1.
If pot is full,
Then do 2 at the same time
Do 3-4-5
When tea is brewed, do 6

Plan 1: do 1.1-1.2-1.3-1.4
When water is boiling, do 1.5

Example 3: Produce a high-level hierarchical task analysis showing how you would find information on a Website. Assume the site has a search facility as well as normal links.



Advantages

- HTA is a simple and flexible method that does not depend on a methodological context.
- HTA enables the representation of a task hierarchy that could be further detailed.

- Although HTA is task oriented and to some extent user oriented it still maintains a strong relationship with traditional software engineering.
- HTA provides information, inefficiencies in tasks that can be used for developing product requirements.

Disadvantages

- There are no strict rules for creating an HTA diagram so different analysts will generate inconsistent hierarchies at varying levels of detail.
- HTA requires both training and experience. It is not a tool that can be applied immediately.
- HTA is not a predictive tool. It focuses on existing tasks.
- HTA diagrams can become quite complex.

Engineering Task Models and CTT
Problems with HTA

- HTA is easy to understand, therefore it can serve as a good starting point for modeling tasks. However, HTA is not very helpful when implementation comes into the picture
- In order to implement something, we need to specify it in an unambiguous way
  –The HTA lacks the rigor and formalism required for such specification
- Engineering task models are more useful as they can be specified formally

Engineering Task Models –Characteristics

- Engineering task models should have flexible and expressive notations, which are able to describe clearly the possible activities
  –Notations should be sufficiently powerful to describe interactive and dynamic behaviors
  –Notations should be readable so that they can also be interpreted by people with little formal background

- An engineering task model should have systematic methods to support the specification, analysis, and use of task models in the design
  –Otherwise, it will be difficult to use the knowledge gained from task analysis
  –Such methods can also be incorporated in tools aiming to support interactive designers

- Another characteristics of an engineering task model is that, it should have support for the reuse of good design solutions to problems that occur across many applications
  –This is especially relevant in industrial context, where developers often have to design applications that address similar problems

- Finally, it is also important that engineering task models make automatic tools available to support the various phases of the design cycle.
  –Tools should have intuitive representations and provide information useful for the logical activities of designers

Concur Task Tree (CTT)

- CTT is an engineering approach to task modeling
- A CTT consists of tasks and operators
  –Operators are used to depict temporal relationships between tasks

- The key features of a CTT are
  –Focus on activities that users aim to perform
  –Hierarchical structure
  –Graphical syntax
  –Rich set of temporal operators

CTT –Task Categories

In CTT, four task categories are defined:

- **User task:** these are tasks that represent only internal cognitive activity of a user, such as selecting a strategy to solve a problem
  –It is graphically depicted with the symbol
  –Can have subtypes such as planning, comparing, problem solving …

- **Interaction task:** these are user actions with possibility of immediate system feedback, such as editing a diagram
  –It is graphically depicted with the symbol
  –Can have subtypes such as selection, edit, control …

- **Application task:** these refer to tasks performed by the system only, such as generating a query result
  –It is graphically depicted with the symbol
  –Can have subtypes such as overview, comparison, locate, grouping, and processing feedback…

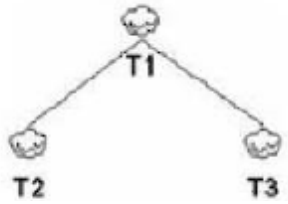
- **Abstract task:** these refer to tasks whose subtasks are of different types (e.g., one user task and one application task) or the task type is not yet decided
  –It is graphically depicted with the symbol

- CTT provides facilities to take care of more task characteristics, namely iterative tasks and optional tasks
  –An iterative task T is denote by **T ***

  –An optional task T is denoted by **[T]**

CTT –Hierarchy

- CTT supports hierarchical task representation. Task at the same level can represent different options or same abstraction level to be performed
- Example: in order to do T1, you have to perform T2 and/or T3

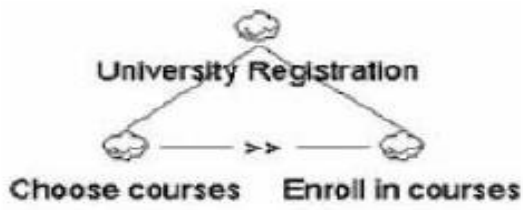


- Note that in CTT, hierarchy does not imply sequence
- In order to represent sequence, the tasks have to be modeled at the same level in a left-to-right manner

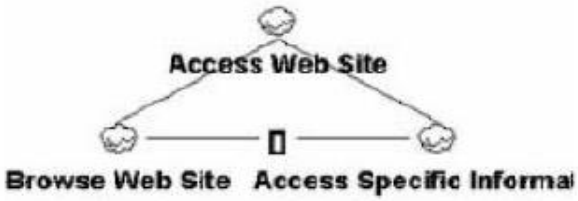
CTT –Temporal Operators

There are eight temporal operators defined in CTT

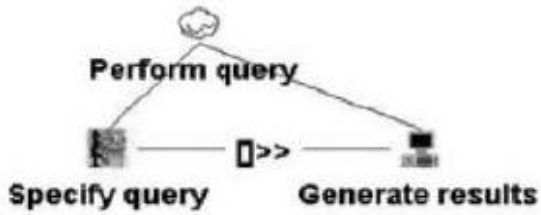
- **Enabling operator (>>):** if two tasks T1 and T2 are related by the operator as T>>T2, it means that T2 cannot occur before T1
  Example: you cannot enroll in a course unless you select the course



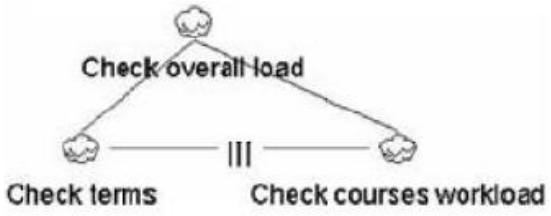
- **Choice operator ([ ]):** if two tasks T1 and T2 are related by the operator as T[ ]T2, it means that both T1 and T2 are enabled, however, only one can be performed at a time
  Example: you can either browse a web site or follow a link for details



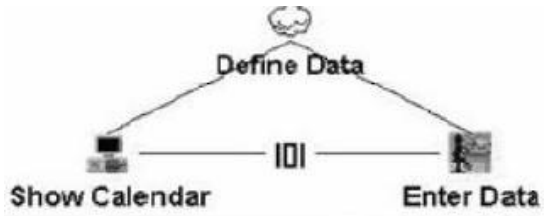
- **Enabling with information passing operator ([ ]>>):** if two tasks T1 and T2 are related by the operator as T[ ]>>T2, it means that T2 can't be performed before T1 and depends on the results of T1
  Example: a system can generate result only after user specifies query. The result depends on the query



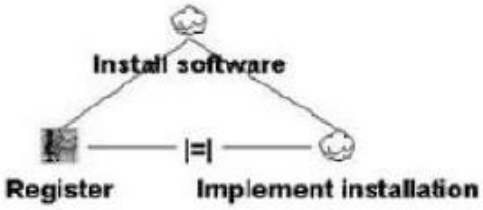
- **Concurrent operator (|||):** if two tasks T1 and T2 are related by the operator as T|||T2, it means that T1, T2 can be performed at any time, in any order
  Example: to check the overall course load, we need to check the semester as well as the course workload



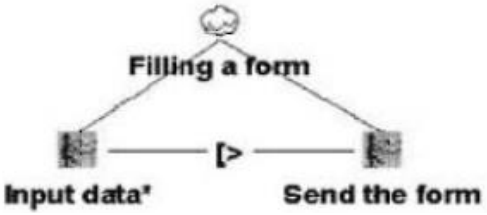
- **Concurrent communicating operator (|[ ]|):** if two tasks T1 and T2 are related by the operator as T|[]|T2, it means that T1, T2 can be performed concurrently and can exchange information
  Example: an onscreen calendar highlighting dates being entered by the user



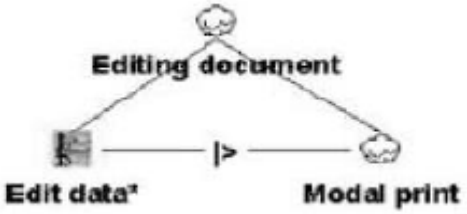
- **Task independence operator (|=|):** if two tasks T1 and T2 are related by the operator as T|=|T2, it means that T1, T2 can be performed independent to one another, however, when one starts, it has to finish before the other can start
  Example: when installing a new software, you can either register and then install or vice-versa



- **Disabling operator ([>):** if two tasks T1 and T2 are related by the operator as T [>T2, it means that T1 (usually iterative) is completely interrupted by T2
  Example: a user can iteratively input data in a form until it is sent



- **Suspend-resume operator (|>):** if two tasks T1 and T2 are related by the operator as T|>T2, it means T1 can be interrupted by T2. When T2 ends, T1 can resume from where it stopped
  Example: editing some data and then printing it, assuming the two can't be performed together



CTT –Other Features

- Tasks in CTT are used to manipulate Objects
- Two types of objects defined

  –Perceivable (user interface) objects –these refer to output objects for presenting information (e.g., windows, tables, graphs) or items which users can interact with (e.g., menus, icons, windows)
  –Application (domain) objects: these are entities that belong to the application domain

- Information concerning application objects needs to be mapped onto perceivable objects to be presented to the user
- Examples of application objects include an order in a business application or a flight in an air traffic control application

- Multiple user interface objects can be associated with a domain object
  —For example, temperature (a domain object) can be represented by a bar-chart (an user interface object) or a textual value (another user interface object)

- Each object can be manipulated by one/more tasks
- Tasks can have other information as attribute (e.g., frequency, informal description, estimated performance time etc.)

CTT –Advantage

- Formal notations help to check for completeness in specification (e.g. each non-basic task has at least two children).
- It allows us to compare two models in terms of no of tasks, no of basic tasks, allocation of tasks, no of instances of temporal operators, the structure of the task models (number of levels, maximum number of sibling tasks etc.)

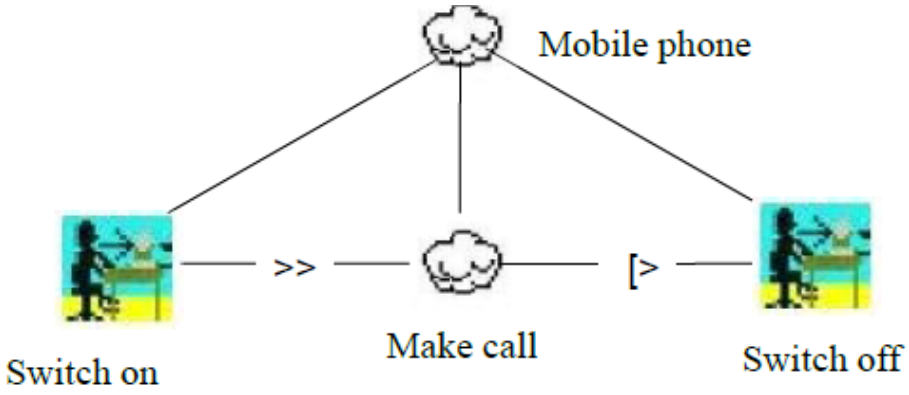
CTT Example:

Suppose we want to model the user tasks in operating a mobile phone. Further assume that we are interested in modeling the tasks related to making a phone call only (not taking photos or listening to music!)

- The questions of interest are,
  –What are the tasks?
  –What are the task types (user tasks, interaction task etc.)?
  –How the tasks are related (what temporal operators are applicable between tasks)?

- First thing to note is that there are three sequential tasks
  –We first switch-on the phone (assuming the phone is always switched-off after making a call)
  –Then we perform the tasks necessary to make phone call
  –Finally we switch-off the phone
- Switching on/off the phone are interaction tasks
- The type of the subtasks necessary to perform a phone call are not of single types (as we shall see). Hence it's an abstract task
- Phone call can't take place before the phone is switched on. Thus, these two are related by the enabling operator (>>)
- Phone can be switched off any time during a call. So, these two tasks are related by a disabling operator ([>)

- **The top level structure of the CTT therefore looks like the following**



- Now, let us determine the subtasks for making a call
  –We first have to select the number of the person to be called (task: select number)
  –Then, we dial the number using a "make call" button (task: dial number)

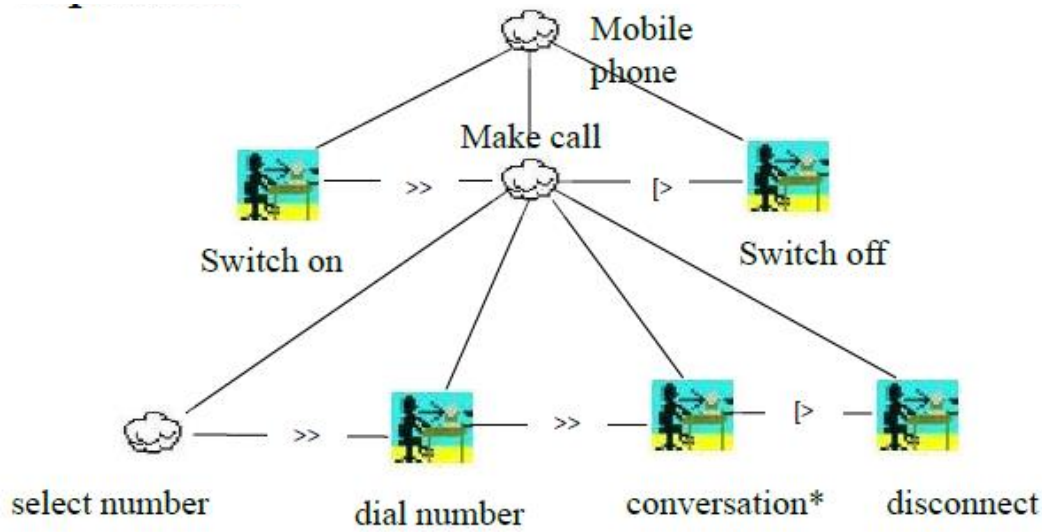
- What are the task types?
- Now, let us determine the subtasks for making a call
  –Once the called person picks up the phone, we start conversation (task: conversation)
  –Finally, at the end, we disconnect (task: disconnect)

- The "select number" tasks involve subtasks of different types (as we shall see). So, it is an abstract task
- "Dial number" is clearly an interaction task
- "Conversation" is an interaction task. Moreover, it is a repetitive task if we assume that a conversation represents each pair of utterance (called/caller)-response (caller/called)
- The remaining task, namely "disconnect" is also interaction task

- Next, we need to find out the relationship between these tasks
- It is obvious that "select number" precedes "dial number", which in turn precedes "conversation"
- "disconnect" comes at the end of the sequence of tasks

- Therefore, "select number" and "dial number" should be connected by the enabling operator (>>). The same operator should apply between "dial number" and "conversation"
- Since a call can be disconnected any time during conversation, "conversation" and "disconnect" should be related with a disabling operator ([>)

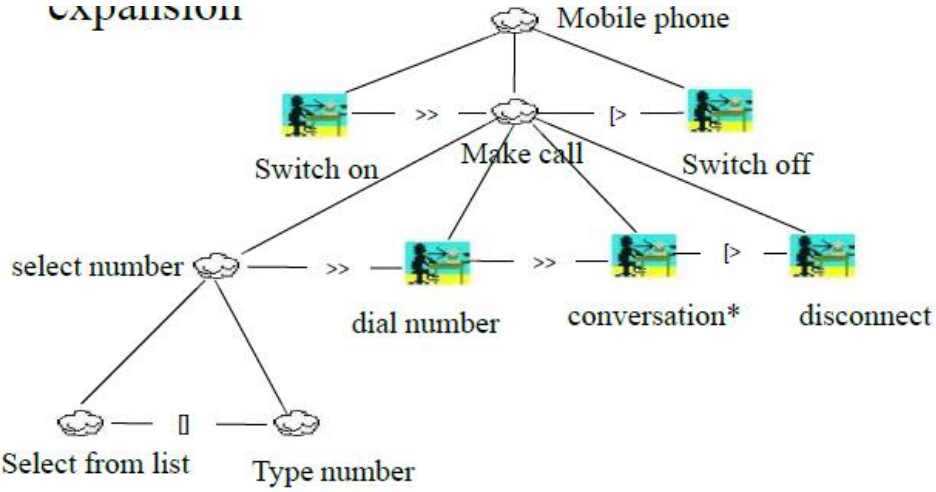
- Thus, the CTT looks like the following after expansion



- The task "select number" can be performed in either of the two ways
  –We can select the number from a contact list (task: select from list)
  –Or we can type the number (task: type number)

- Both the above tasks are abstract type (as they can be further divided into different types of subtasks) and connected by the choice operator ([])

- Thus, the CTT looks like the following after expansion



- The task “select from list” involves four sub tasks
  –First, select the appropriate button/menu option to display the list (task: open list)
  –Next, browse the list elements till you arrive at the name of the called person (task: find name)

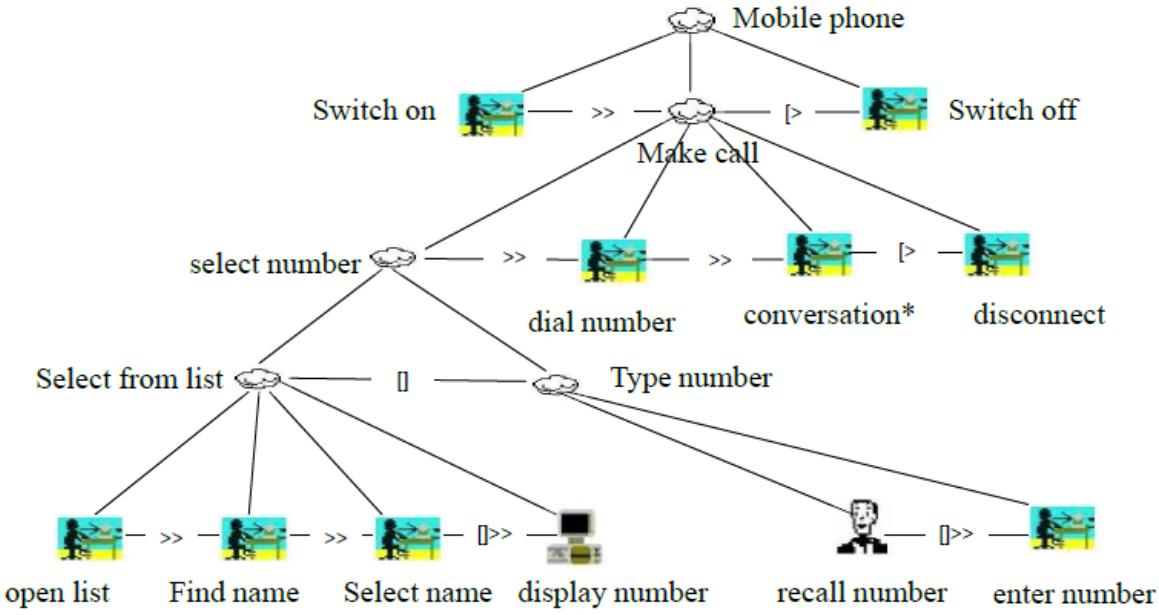
- The task “select from list” involve four sub tasks
  –Then, select the name using appropriate button (task: select name)
  –Finally, once the name is selected, the corresponding number is displayed on the screen (task: show number)

- The first three tasks (display list, find name, select name) are all interaction tasks
- Since these have to be performed in sequence, they are connected by the enabling operator (>>)

- The last task (display number) is of type application task. Moreover, it is related with the last task in the previous sequence (select name) through the enabling with information passing operator ([]>>)

- The task “type number” involves two subtasks
  –We first need to recall the number (task: recall number). Clearly, this is a user task
  –Then we need to actually type the numbers (task: enter number). This is clearly an interaction task
- The two tasks are related through the enabling with information passing operator ([]>>)

- Thus, the CTT for the entire task looks like the following



Dialog Design

- One important aspect of HCI is the dialog, which is the interaction that takes place between a human user and the computer

Dialog

- A dialog refers to the structure of the interaction
- Dialog in HCI can be analyzed at three levels
  –Lexical: at this level, details such as the shape of icons, actual keys pressed etc. are dealt with
  –Syntactic: the order of inputs and outputs in an interaction are described at this level
  –Semantic: the effect a dialog has on the internal application/data is the subject matter at this level

Dialog Representation

- We need (formal) techniques to represent dialogs, which serves two purposes
  –It helps to understand the proposed design better
  –Formal representation makes it possible to analyze dialogs to identify usability problems (e.g., we can answer questions such as "does the design actually support undo"?)

There are several formalism that we can use to represent dialogs

- The state transition networks (STN)
- The state charts
- The (classical) Petri nets

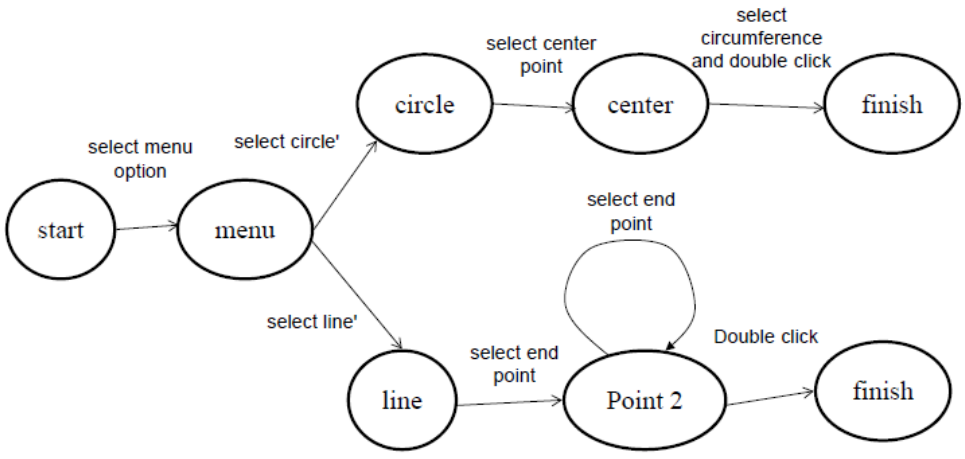
State Transition Network (STN)

- STNs are the most intuitive among all formalisms
- It assumes that a dialog essentially refers to a progression from one state of the system to the next in the system state space (in fact this assumption holds for all formalisms that we shall discuss)

- **The syntax of an STN is simple and consists of the following two entities**
  –Circles: a circle in a STN refers to a state of the system, which is labeled (by giving a name to the state)
  –Arcs: the circles are connected with arcs, each of which refers to the action/event (represented by arc labels) that results in the system making a transition from the state where the arc originates to the state where it terminates.

Example. Suppose, we are using a drawing interface that allows us to draw lines and circles, by choosing appropriate menu item. To draw a circle, we need to select a center and circumference. A line can be drawn by selecting points on the line. How can we model this dialog using an STN?

- We shall have a "start" state
- From this "start" state, we shall go to a "menu" state, where we are shown the menu options. If we select the circle option, we go to a "circle" state. Otherwise, we select the "line" option and go to the "line" state
- While at the "circle" state, we select a point as the circle center (through mouse click, say), which takes us to the "center" state
- In the "center" state, we select the circle periphery (through mouse movement, say) and double click to indicate the end of input (the "finish" state). At this stage, the circle is displayed
- While at the "line" state, we select a point as the beginning of the line (through mouse click, say)
- Then, we select another point to denote the last point on the line and transit to "point 2". At this stage, a line is displayed between the two points
- We can select another point, while at "point 2" to draw another line segment between this point and the point last selected. We can actually repeat this as many times as we want, to draw line of arbitrary shape and size
- When we perform a double click, it indicates the end of input and the dialog comes to the "finish" stage



STN –Pros and Cons

Pros

- –Intuitive
- –Easy to understand

Cons

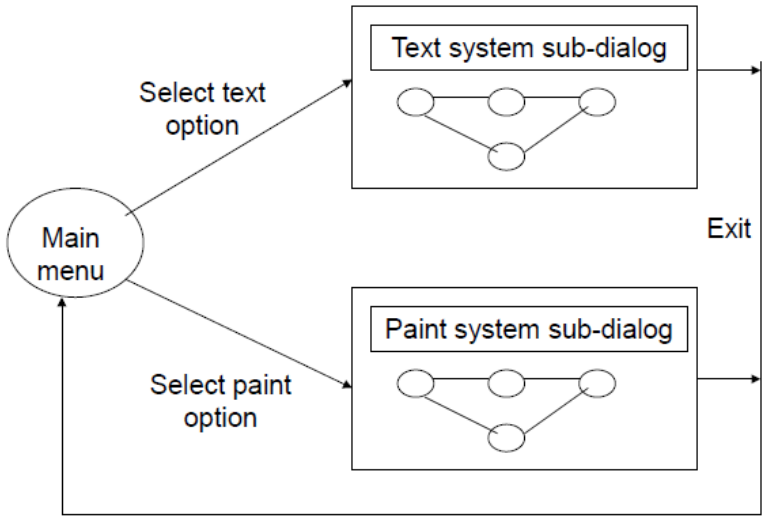
- –Good for simple systems
- –Quickly becomes messy as the number of states/arcs grow

How to Model Complex Dialogs

- Hierarchical STNs provide a way to manage complex dialogs
- Here, we divide the dialog into sub-dialogs
- Each sub-dialog is modeled with STNs
- Upper level STNs are designed to connect sub-dialogs

Hierarchical STN –Example

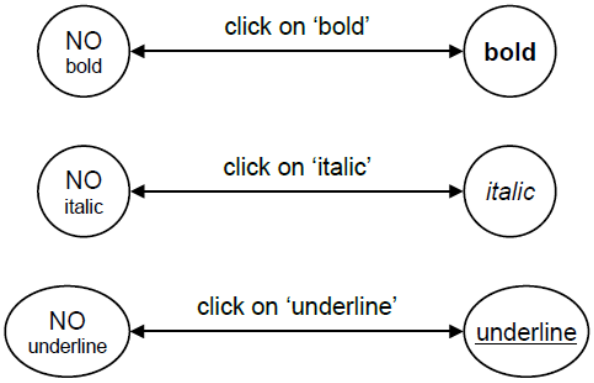
Suppose we want to model the dialog for a menu-based system. There are two menu items, one for a text system and the other for a paint-like system. Each of these systems has its own dialog. For example, the paint system may have the dialog shown in the previous example. We can model the overall dialog as a hierarchical STN, as shown in the next slide

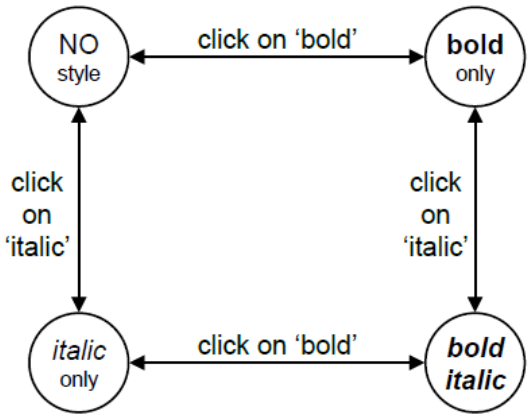


How to Model Complex Dialogs

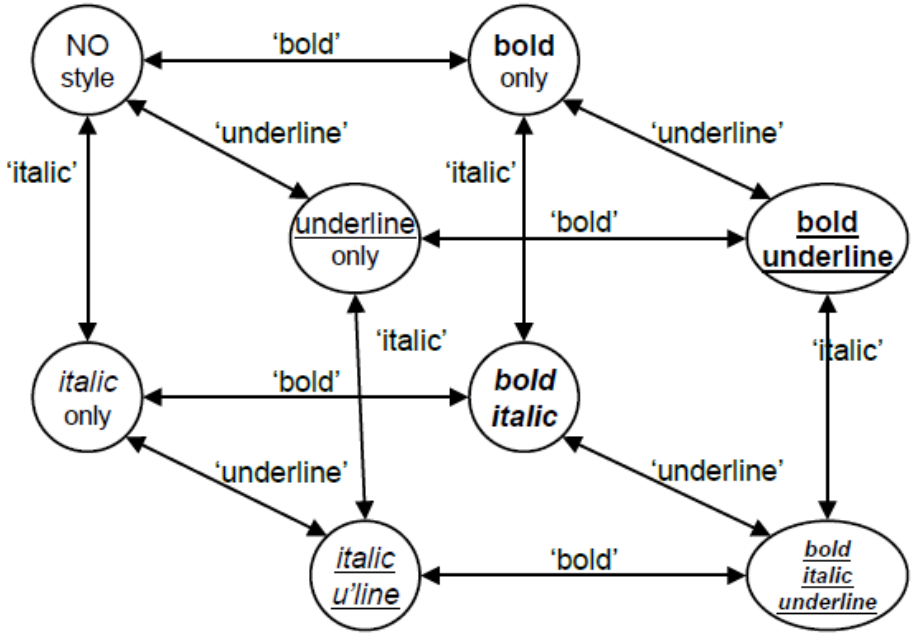
- However, even hierarchical STNs are inadequate to model many "common" dialogs
- For example, consider a text editor that supports three operations: underline, bold and italic. Let us try to model this dialog with an STN, assuming first that we can perform (only) one operation on a piece of text

Modelling Complex Dialogs





- Now suppose we relax the condition "we can perform (only) one operation on a piece of text". Now we can perform two operations together on the same piece of text (e.g., bold followed by italic). How the STN for this new system looks?
- (let us construct the STN for only the dialog involving bold and italic. STN for other pairs will be similar)
- Now suppose we relax the condition further. Now we can perform all the three operations together on the same piece of text. This is a fairly common scenario and supported by all text editors. Let us see how the STN for this new system looks.



- As you can see, the STN has become very complex with too many states and arcs
- This is because we are trying to model activities that occur on the same object or at the same time. Such behaviors are known as "concurrent behaviors"
- STNs are not very good at modeling concurrent behaviors, which are fairly common in dialogs that we encounter in HCI

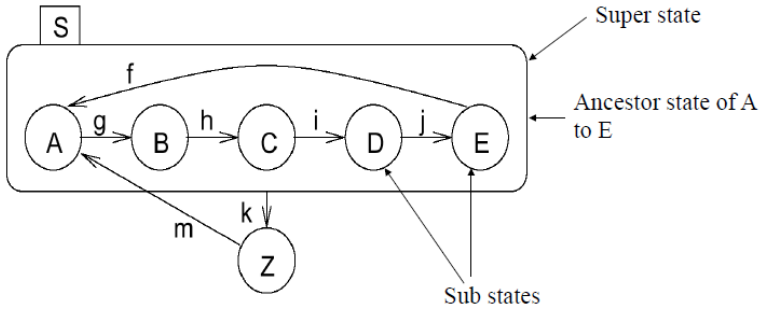
State Charts

- Proposed by David Harel (1987) to represent complex reactive systems
- Extends finite state machines (FSM)
  –Better handle concurrency
  –Adds memory, conditional statements to FSM
- Simplifies complex system representation (states and arcs) to a great extent

Definitions

- **Active state:** the current state of the underlying FSM
- **Basic states:** states that are not composed of other states
- **Super states:** states that are composed of other states

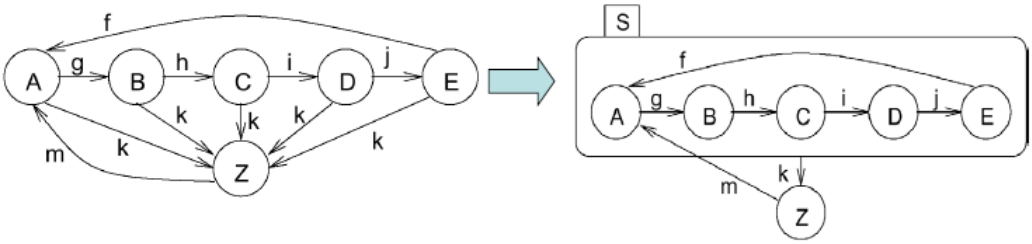
- For each basic state b, the super state containing b is called the ancestor state
- A super state is called OR super state if exactly one of its sub states is active, whenever it is active



Super State Advantage

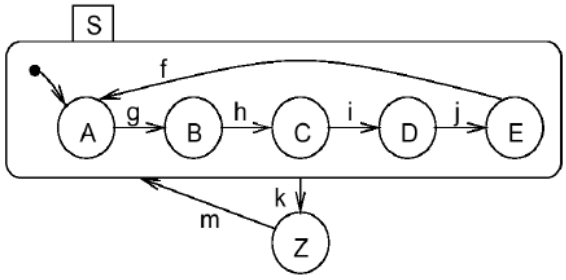
Clustering

It allows us to represent complex FSM in a nice way, by clustering states



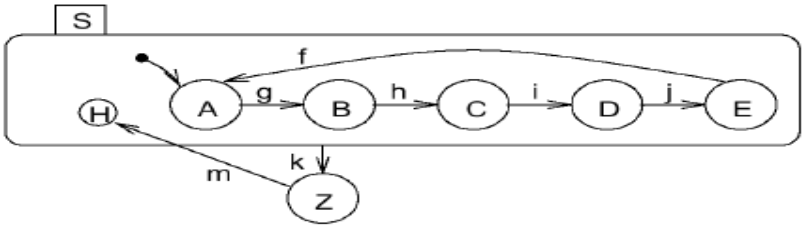
Default State Mechanism

- Indicates the sub state entered whenever super state is entered –represented using a filled circle –Not a state by itself

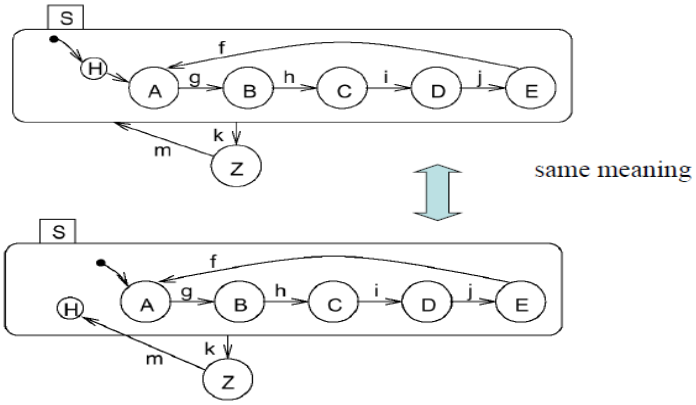


History Mechanism

- For input m, S enters the state it was in before S was left
  –If S is entered for the very first time, the default mechanism applies
  –History and default mechanisms can be used hierarchically

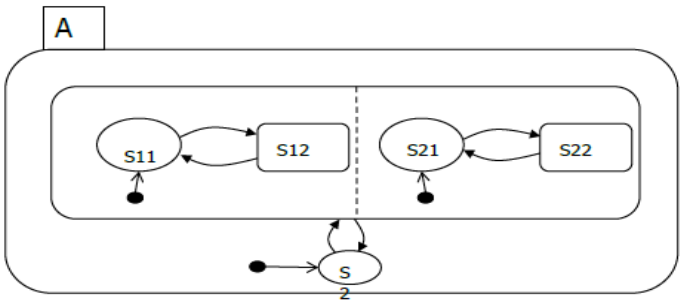


Combining History and Default State Mechanism



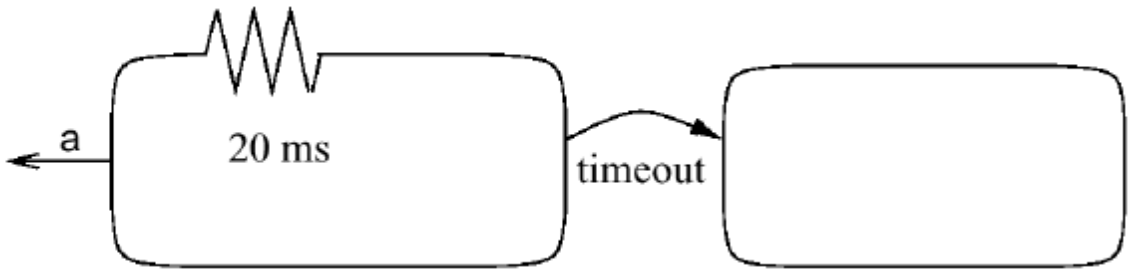
AND Super State

- State Charts supports concurrency using the notion of the AND super states
  –In AND super states, the FSM is active in all (immediate) sub states simultaneously



Timing Constraints

- State Chart supports delay/timeout modeling –using special edges

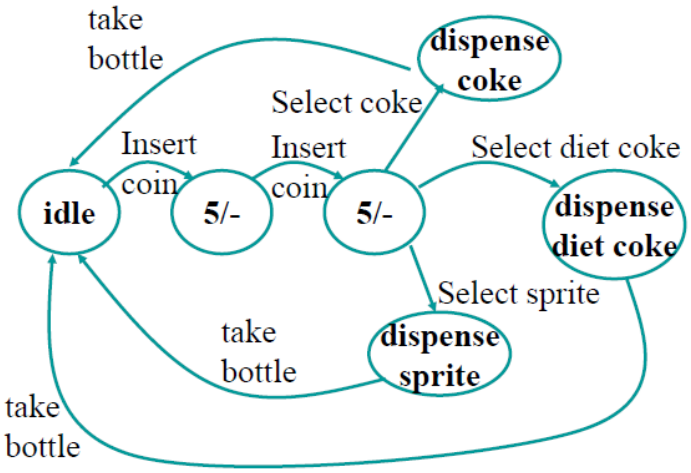


- If event a does not happen while the system is in the left state for 20 ms, a timeout will take place.

Example: Coke Machine

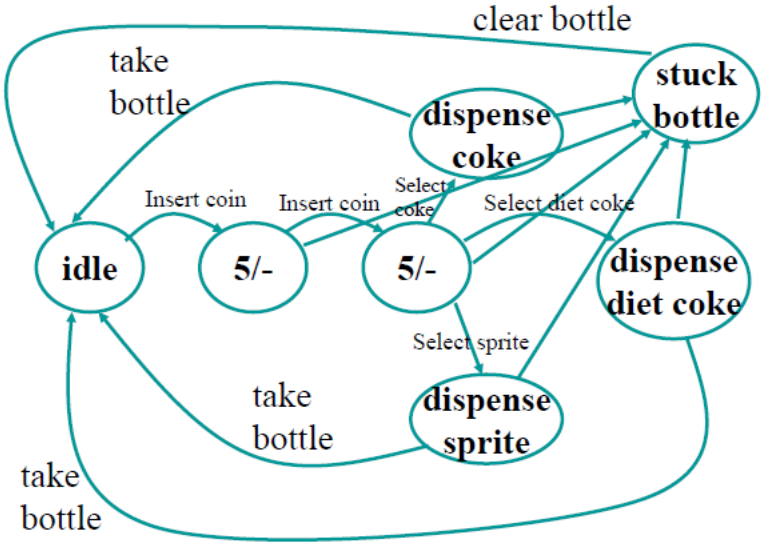
- Suppose you have a coke vending machine:
  –When turned on, the machine waits for money
  –When Rs. 5/-coin is deposited, the machine waits for another Rs. 5/-coin
  –When the second coin is deposited, the machine waits for a selection
  –When the user presses "COKE," a coke is dispensed
  –When the user takes the bottle, the machine waits again
  –When the user presses either "SPRITE" or "DIET COKE," a Sprite or a diet Coke is dispensed
  –When the user takes the bottle, the machine waits again

Represent this behavior using FSM
Coke Machine Version 1.0

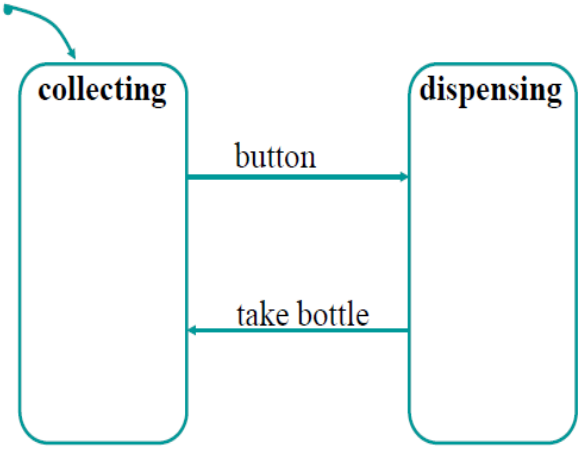


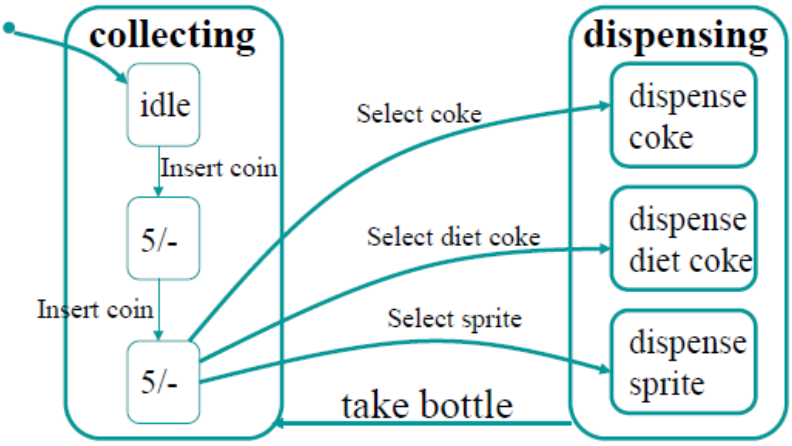
Coke Machine, Version 2.0

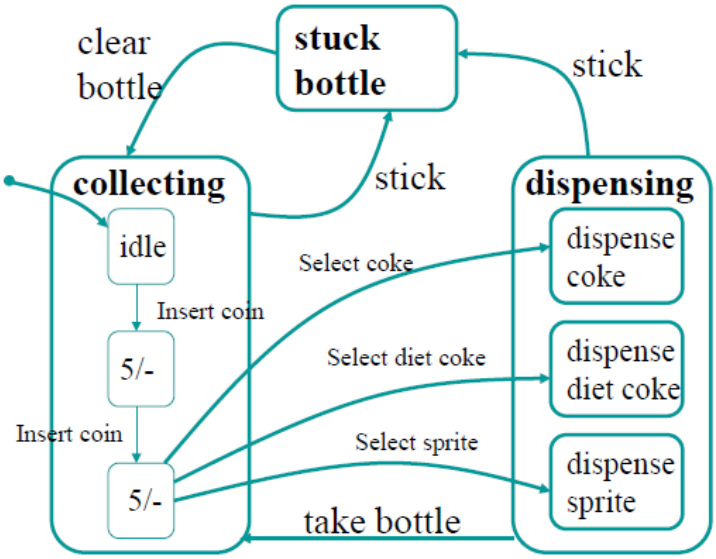
- Let's include some more features in the machine
- The Bottles can get stuck in the machine
  –An automatic indicator will notify the system when a bottle is stuck
  –When this occurs, the machine will not accept any money or issue any bottles until the bottle is cleared
  –When the bottle is cleared, the machine will wait for money again

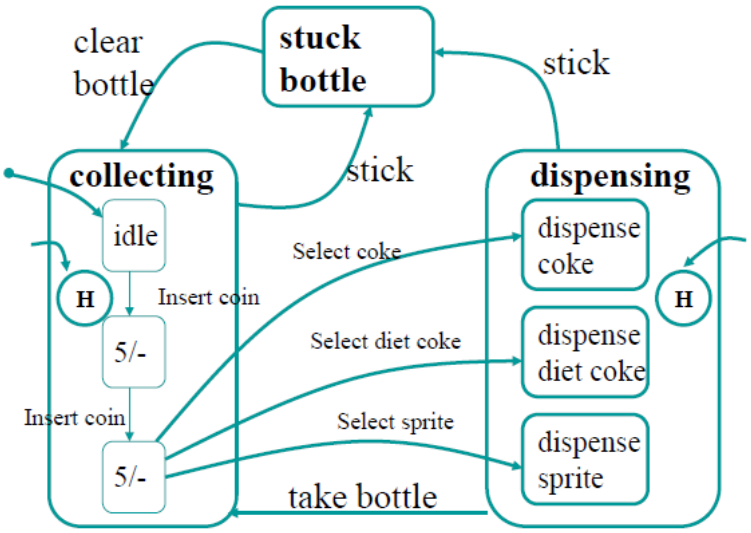


State Chart Construction: Bottle Dispenser









Petri Nets

- The formalism was first proposed by Carl Adam Petri (1962, PhD thesis)
- It is a simple model of dynamic behavior
  –Just four elements are used to represent behavior: places, transitions, arcs and tokens
  –Graphical and mathematical description for easy understanding
  –Formal semantics allow for analysis of the behavior

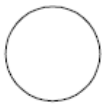
Elements of PN

- **Place:** used to represent possible states of the systems.
- **Transition:** used to represent events or actions which cause the change of state
- **Arc:** used to represent causal relations/ connect a place with transition or vice versa.
- **Token:** elements subject to change

The state (space) of a process/system is modeled by places and tokens and state transitions are modeled by transitions

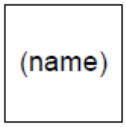
Elements of PN: Notation

- A place is represented by a circle



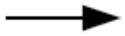
place

- Transitions are represented by squares/rectangles



transition

- Arc are represented by arrows



arc (directed connection)

- Tokens are represented by small filled circles

token

Role of a Token

- **Tokens can play the following roles**
  –A physical object, for example a product, a part, a drug, a person
  –An information object, for example a message, a signal, a report
  –A collection of objects, for example a truck with products, a warehouse with parts, or an address file
  –An indicator of a state, for example the indicator of the state in which a process is, or the state of an object
  –An indicator of a condition: the presence of a token indicates whether a certain condition is fulfilled

Role of a Place

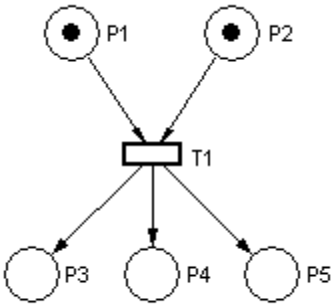
- **A place in a PN can represent the following**
  –A type of communication medium, like a telephone line, a middleman, or a communication network
  –A buffer: for example, a depot, a queue or a post bin
  –A geographical location, like a place in a warehouse, office or hospital
  –A possible state or state condition: for example, the floor where an elevator is, or the condition that a specialist is available

Role of a Transition

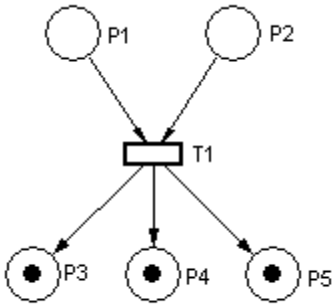
- **A transition can be used to represent things such as**
  –An event (e.g., starting an operation, the switching of a traffic light from red to green)
  –A transformation of an object, like adapting a product, updating a database, or updating a document
  –A transport of an object: for example, transporting goods, or sending a file

PN Construction Rules

- Arcs have capacity 1 by default; if other than 1, the capacity is marked on the arc.
- Places have infinite capacity by default, and transitions have no capacity, and cannot store tokens at all.
- A transition is enabled when the number of tokens in each of its input places is at least equal to the arc weight going from the place to the transition.
- An enabled transition may fire at any time. When fired, the tokens in the input places are moved to output places, according to arc weights and place capacities.

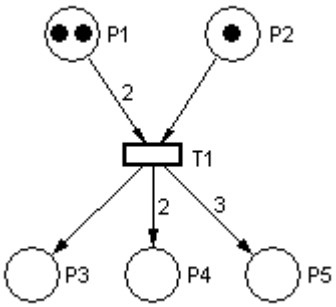


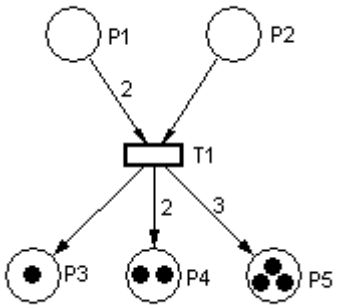
Enabled.ready to file



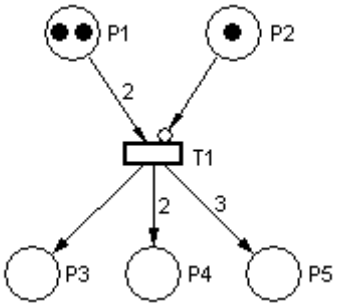
Firing complete

- When arcs have different weights, we have what might at first seem confusing behavior. Here is a similar net, ready to fire: and here it is after firing:





- Change of state is denoted by a movement of *token(s)* (black dots) from place(s) to place(s); and is caused by the *firing* of a transition.
- The firing represents an occurrence of the event or an action taken.
- A transition is *firable* or *enabled* when there are sufficient tokens in its input places.
- After firing, tokens will be transferred from the input places (old state) to the output places, denoting the new state.
- If the arc weights are all the same, it appears that tokens are moved across the transition. If they differ, however, it appears that tokens may disappear or be created. That, in fact, is what happens; think of the transition as removing its enabling tokens and producing output tokens according to arc weight.
- A special kind of arc, the inhibitor arc, is used to reverse the logic of an input place. With an inhibitor arc, the absence of a token in the input place enables, not the presence:



Remarks

- Firing is atomic (i.e., it always completes after start)
- Non-determinism: multiple transitions may be enabled, but only one fires at a time
- The state of the reactive system is represented by the distribution of tokens over places (also referred to as marking)

Modeling Power:

The typical characteristics exhibited by the activities in a dynamic event-driven system, such as concurrency, decision making, synchronization and priorities, can be modeled effectively by Petri nets.

1. **Sequential Execution.** In Figure (a), transition t2 can fire only after the firing of t1. This imposes the precedence constraint "t2 after t1." Such precedence constraints are typical of the execution of the parts in a dynamic system. Also, this Petri net construct models the causal relationship among activities.

2. **Conflict.** Transitions t1 and t2 are in conflict in Figure (b). Both are enabled but the firing of any transition leads to the disabling of the other transition. Such a situation will arise, for example, when a machine has to choose among part types or a part has to choose among several machines.

3. **Concurrency.** In Figure (c), the transitions t1, and t2 are concurrent. Concurrency is an important attribute of system interactions. Note that a necessary condition for transitions to be concurrent is the existence of a forking transition that deposits a token in two or more output places.

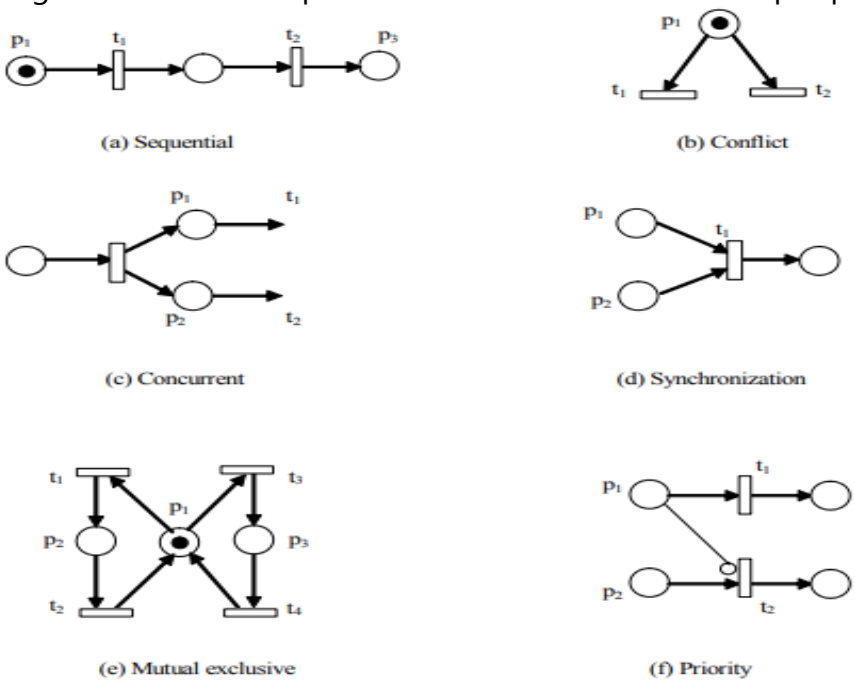


Figure 3 Petri net primitives to represent system features.

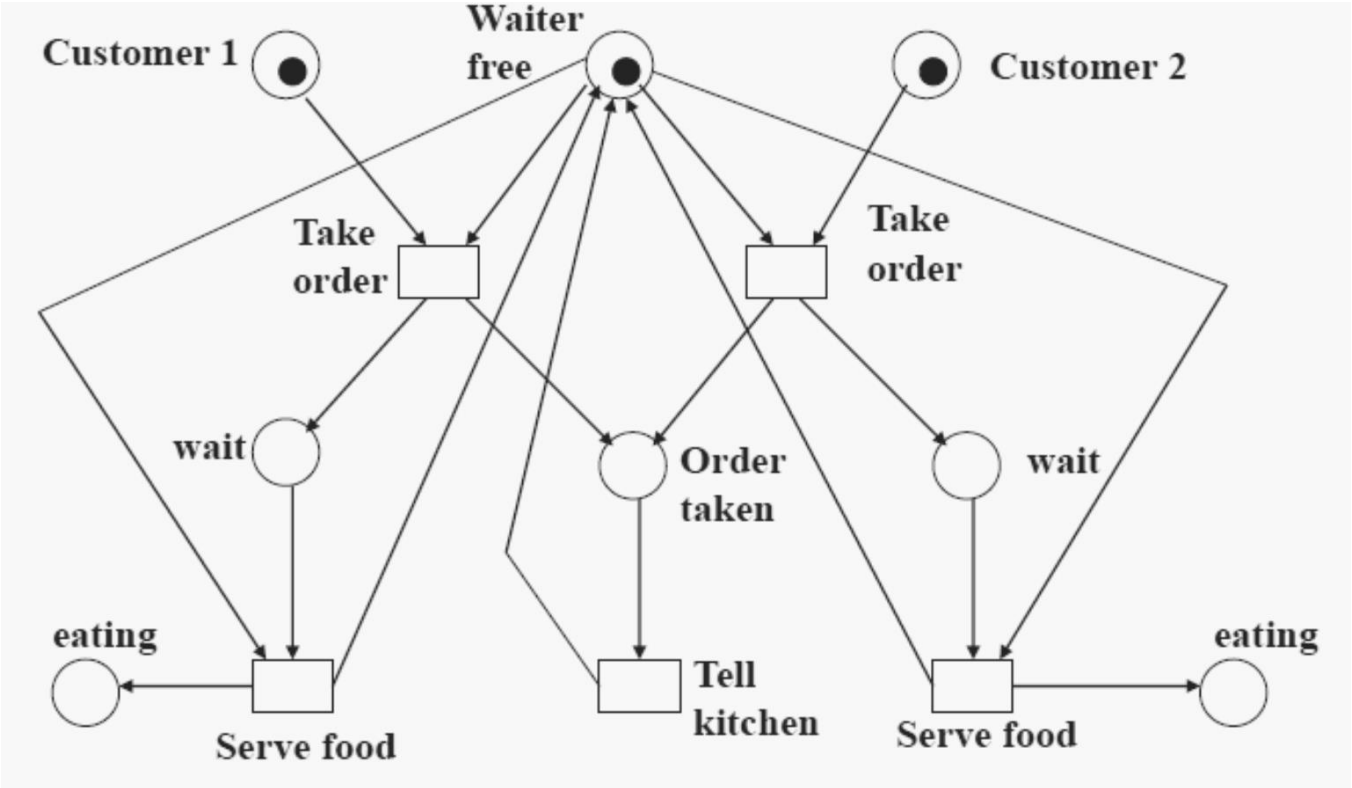
4. **Synchronization.** It is quite normal in a dynamic system that an event requires multiple resources. The resulting synchronization of resources can be captured by transitions of the type shown in Figure (d). Here, t1 is enabled only when each of p1 and p2 receives a token. The arrival of a token into each of the two places could be the result a possibly complex sequence of operations elsewhere in the rest of the Petri net model. Essentially, transition t1 models the joining operation.

5. **Mutually exclusive.** Two processes are mutually exclusive if they cannot be performed at the same time due to constraints on the usage of shared resources. Figure (e) shows this structure. For example, a robot may be shared by two machines for loading and unloading. Two such structures are parallel mutual exclusion and sequential mutual exclusion.

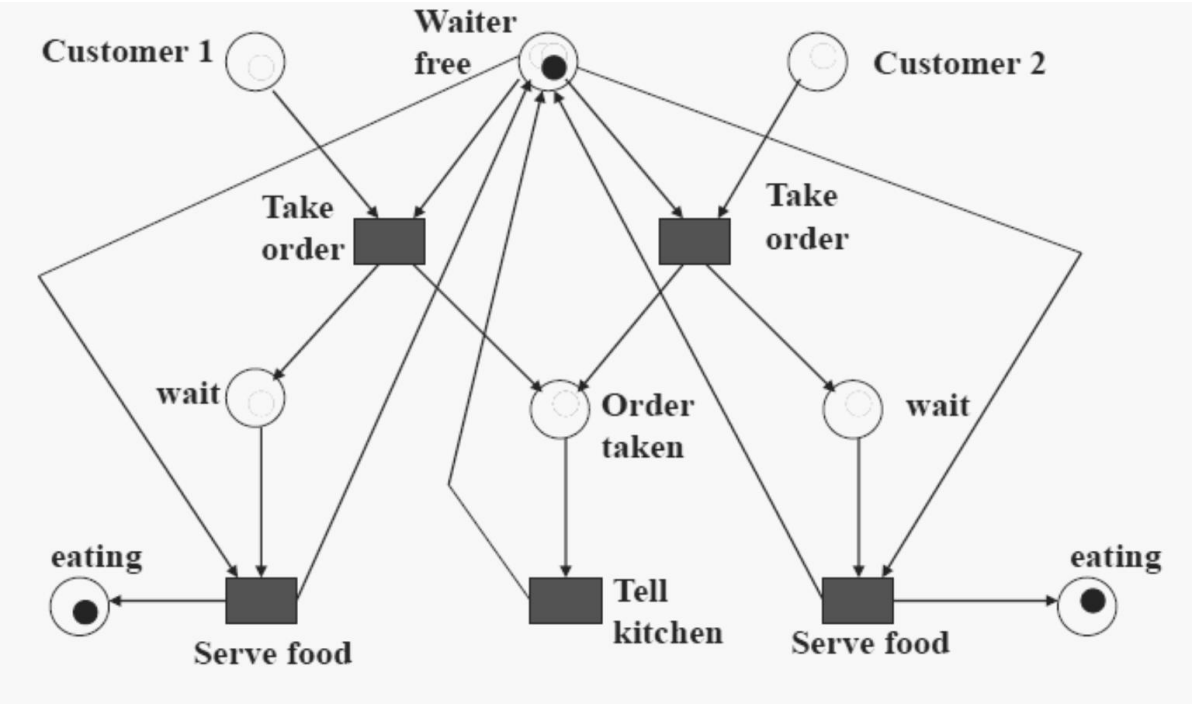
6. **Priorities**. The classical Petri nets discussed so far have no mechanism to represent priorities. Such a modeling power can be achieved by introducing an inhibitor arc. The inhibitor arc connects an input place to a transition, and is pictorially represented by an arc terminated with a small circle. The presence of an inhibitor arc connecting an input place to a transition changes the transition enabling conditions. In the presence of the inhibitor arc, a transition is regarded as enabled if each input place, connected to the 5 transition by a normal arc (an arc terminated with an arrow), contains at least the number of tokens equal to the weight of the arc, and no tokens are present on each input place connected to the transition by the inhibitor arc. The transition firing rule is the same for normally connected places. The firing, however, does not change the marking in the inhibitor arc connected places. A Petri net with an inhibitor arc is shown in Figure (f). t1 is enabled if p1 contains a token, while t2 is enabled if p2 contains a token and p1 has no token. This gives priority to t1 over t2.

Example of Petri nets:

Example1: In a Restaurant (A Petri Net)

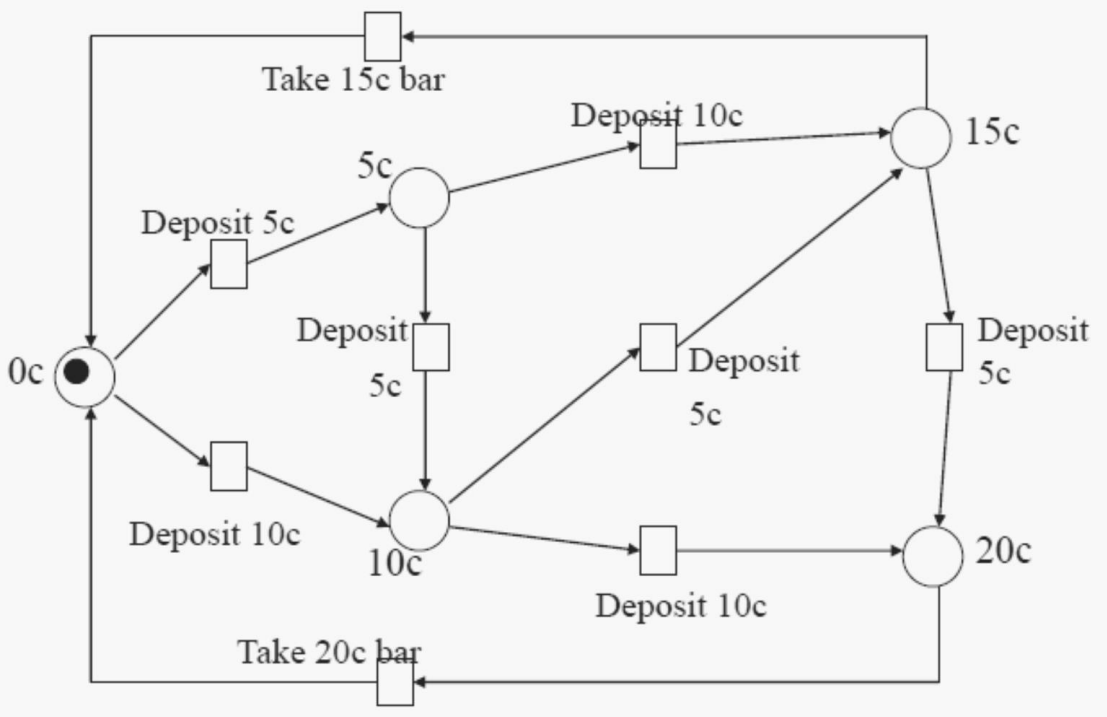


Example: In a Restaurant (Two Scenarios)
Scenario 1:
–Waiter takes order from customer 1; serves customer 1; takes order from customer 2; serves customer 2.
Scenario 2:
–Waiter takes order from customer 1; takes order from customer 2; serves customer 2; serves customer 1.



Example 2: Vending Machine
- The machine dispenses two kinds of snack bars –20c and 15c.
- Only two types of coins can be used –10c coins and 5c coins.
- The machine does not return any change.
  (C= cent)

Example: Vending Machine (A Petri net)



Example: Vending Machine (3 Scenarios)
Scenario 1:
–Deposit 5c, deposit 5c, deposit 5c, deposit 5c, take 20c snack bar.
Scenario 2:
–Deposit 10c, deposit 5c, take 15c snack bar.
Scenario 3:
–Deposit 5c, deposit 10c, deposit 5c, take 20c snack bar.