UNIT–5 : Task Modelling and Dialog Design (PYQ Solution)

PART–A Answers (≈70 Words Each)

Q1. State Hierarchical Task Analysis (HTA). / Describe the key components of an HTA model. (2022, 2024)

HTA describes a task as a hierarchy of goals broken into sub-tasks, with plans that say the order and conditions of doing them. Key components: goals/tasks (numbered nodes), sub-tasks (redescription), plans (sequence, choice, iteration, parallel) and a stopping rule.

Q2. Define Concur Task Tree. (2024, Feb 2024)

A Concur Task Tree (CTT) is a hierarchical task model in which tasks (user, interaction, application, abstraction) are linked by temporal operators such as enabling (>>), choice ([]), interleaving (|||) and disabling ([>), and it supports concurrent tasks.

Q3. What are the benefits and limitations of using FSMs in dialog design? (2024)

Benefits: simple and easy to understand, can be checked for dead ends and unreachable states, directly implementable.
Limitations: state explosion in large dialogs, no concurrency, no hierarchy, difficult to represent data.

PART–B Answers (≈250 Words Each)

Q1. Write short note on - (a) Engineering task models (b) Concur Task Trees (CTT). (2022)

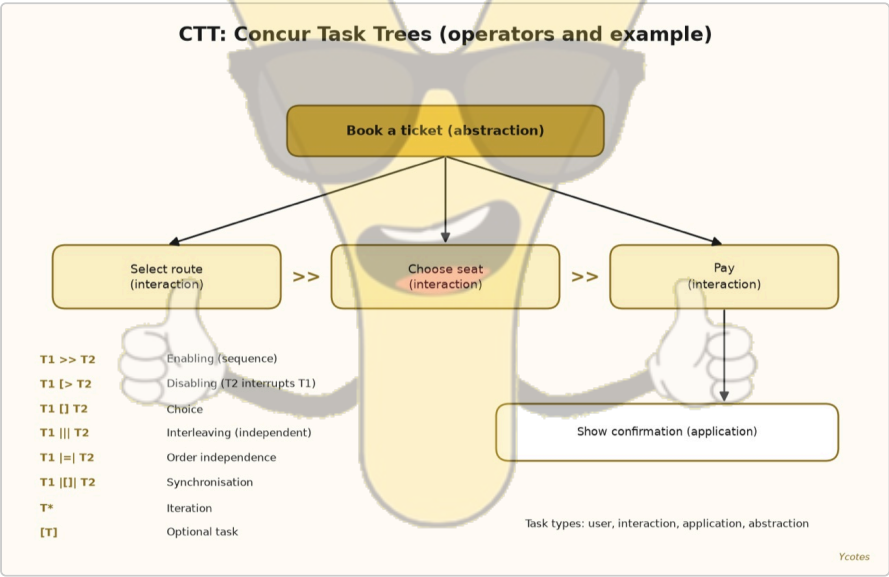
(a) Engineering task models

These are formal descriptions of user tasks (HTA, GOMS, CTT, TKS) used by engineers to analyse tasks and to derive and check the user interface.

- Capture tasks, sub-tasks, objects and the order of actions
- Allow tools to validate the design and generate screens
- Help in task allocation between user and system and in estimating performance

(b) Concur Task Trees (CTT)

CTT (Paterno) represents tasks as a tree with four task types: **user**, **interaction**, **application** and **abstraction**. Siblings are linked by temporal operators.



Operator	Meaning
T1 >> T2	Enabling: T2 after T1
T1 [> T2	Disabling: T2 interrupts T1
T1 [] T2	Choice
T1	
T1	=
T*	Iteration
[T]	Optional

Example: *Book a ticket* = Select route >> Choose seat >> Pay, followed by the application task Show confirmation. Advantages: concurrency is modelled and the CTTE tool supports simulation.

Q2. What do you mean by diagrammatic notation and dialog semantics? explain in brief. (Feb 2024)

Diagrammatic notation is the graphical way of drawing a dialog: boxes, circles, arrows and bars, as in state transition diagrams, statecharts and Petri nets. It shows the **structure** of the dialog (syntax) and is easy to read.

Dialog semantics is the **meaning** of the dialog: what each state, event or transition does to the system and the user (for example, pressing Save writes the file and moves to the Done state).

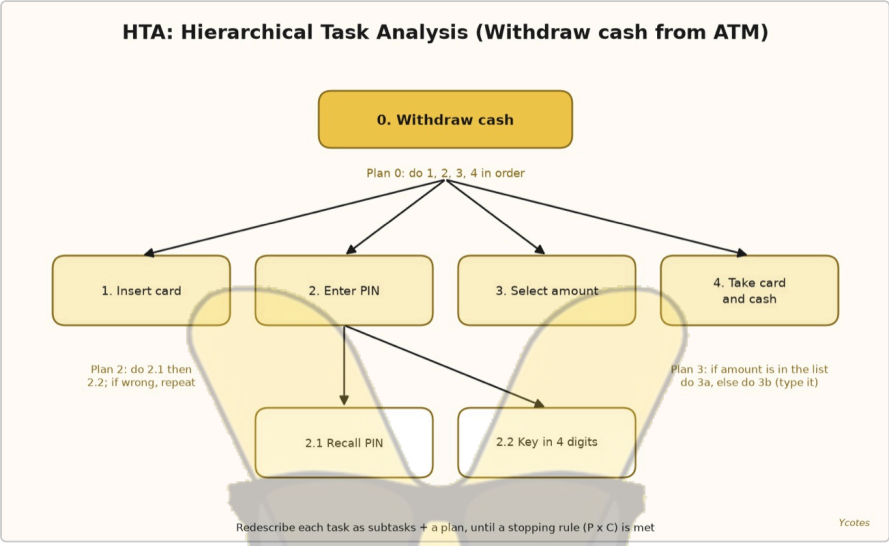
Notation	Semantics
Picture and symbols (syntax)	Behaviour and effect (meaning)
Circles, arrows	"On OK, validate the form and go to next screen"

Both are needed: notation communicates the design, semantics makes it precise so it can be checked and implemented.

Q3. What are the two major techniques that are applied for hierarchical task analysis? Explain. (Feb 2024)

(Assumption: the two techniques are task decomposition and plan specification.)

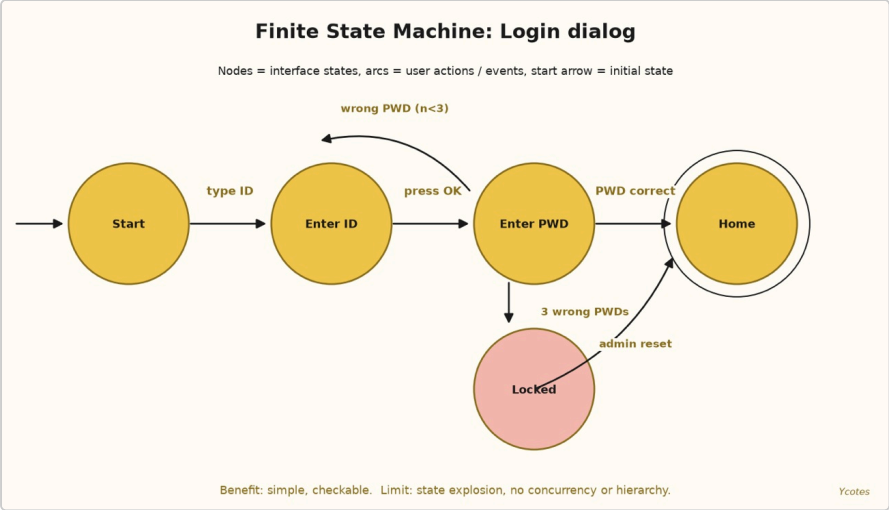
1. **Top-down decomposition (redescription):** the main goal is broken into sub-tasks, and each sub-task is broken again, until a **stopping rule** is met (the probability of failure times the cost of failure is acceptable). Tasks are numbered 0, 1, 2, 2.1 and so on.
2. **Plan specification:** for every parent, a plan states the order and conditions of the sub-tasks: fixed sequence (do 1, then 2), choice (if X do 3a else 3b), iteration (repeat until) or parallel (do 4 and 5 together).



Example: Withdraw cash 0 has sub-tasks 1 Insert card, 2 Enter PIN, 3 Select amount, 4 Take card and cash with Plan 0: do 1 to 4 in order; Plan 2: do 2.1 then 2.2, and repeat if wrong. The result is shown as a hierarchical diagram or a numbered text list.

Q4. Describe the Finite State Machine method and explain how it is used to design and evaluate user interfaces. Discuss the benefits and limitations also. (2024)

An **FSM** is a model with a finite set of **states**, **inputs (events)**, **transitions** and a start (and final) state.



Use in design: each screen or mode of the UI is a state, each user action is an event, and an arrow shows the next state. A state-transition table lists every (state, event) pair:

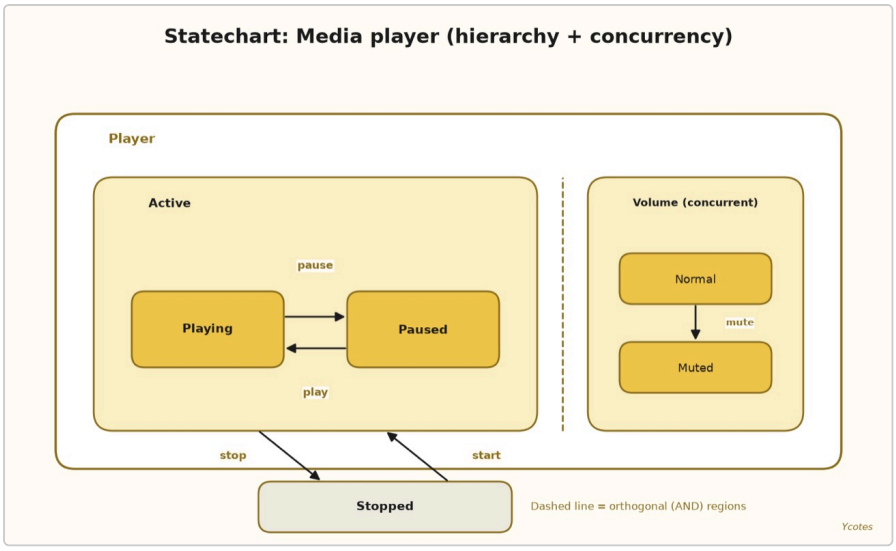
State	Event	Next state
Enter ID	type ID, OK	Enter PWD
Enter PWD	password correct	Home
Enter PWD	wrong, tries < 3	Enter ID
Enter PWD	wrong 3 times	Locked

Use in evaluation: check that every state is reachable, no state is a dead end, every event is handled in each state, and count steps to reach a goal (efficiency).

Benefits: simple, easy to learn, visual, checkable and easy to code.
Limitations: state explosion, no concurrency, no hierarchy, awkward for data and modes, and large diagrams become unreadable.

Q5. Describe the Statechart method and explain how it is used to design and evaluate user interfaces. Discuss the benefits and limitations of using Statecharts. (2024)

Statecharts (Harel) extend FSMs with **hierarchy** (nested states), **concurrency** (orthogonal regions) and broadcast events and history.



Use in design: a window or mode is a super-state containing sub-states. In the media player, *Active* contains *Playing* and *Paused*; an independent *Volume* region (Normal, Muted) runs at the same time; a single *stop* transition leaves Active from any sub-state, which replaces many arrows of a plain FSM.

Use in evaluation: check that every event is handled in each state, that concurrent regions do not conflict and that modes are consistent. The model can be simulated before coding.

Benefits: compact, models hierarchy and concurrency, scales to large UIs, reduces clutter.
Limitations: harder to read and learn than an FSM, semantics are complex, little support for data, and tools are needed.

PART–C Answers (≈500 Words Each)

Q1. Explain introduction to formalism in dialog design and also explain Finite State Machines with suitable example. (2022)

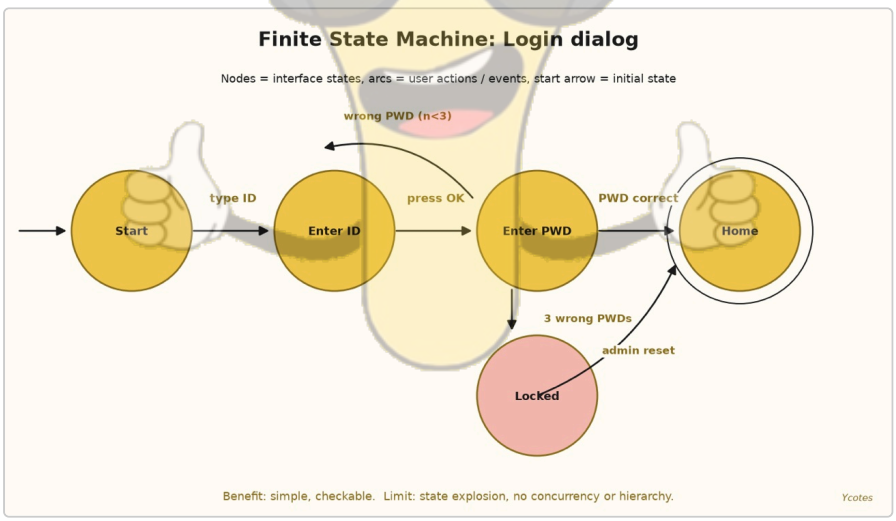
Dialog is the conversation between user and system (sequence of inputs and outputs). A **formalism** is a precise notation for describing it, so that the dialog can be specified without ambiguity, analysed for completeness and consistency, and implemented correctly.

Need: natural-language specifications are vague; formal dialogs allow early checking (dead ends, missing events, unreachable screens) and communicate between designer and programmer.

Types of formalisms: state transition networks (FSM), statecharts, Petri nets, flow charts and grammars (BNF).

Finite State Machines

An FSM has states S, events E, transition function $S \times E \rightarrow S$, a start state and final states.



Example: login dialog. States: Start, Enter ID, Enter PWD, Home (final), Locked.

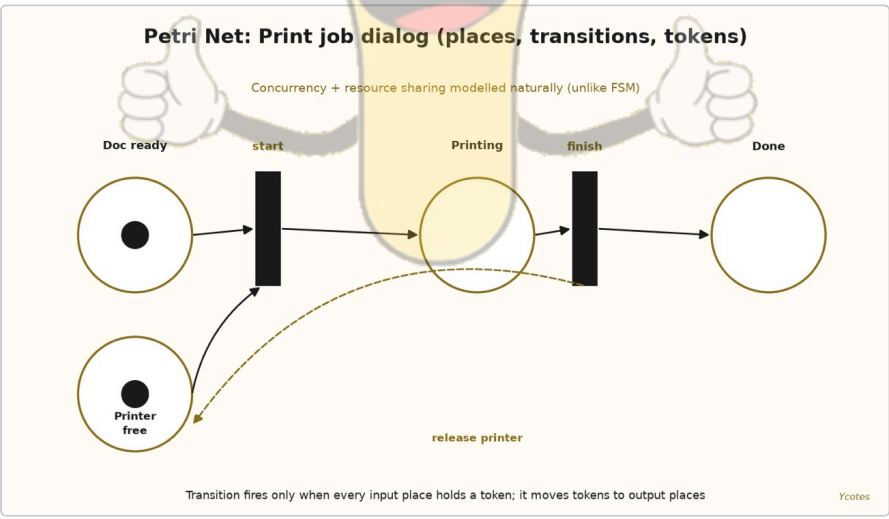
- Start to Enter ID on opening the screen
- Enter ID to Enter PWD on OK
- Enter PWD to Home when the password is correct
- Enter PWD back to Enter ID on a wrong password (fewer than 3 tries)
- Enter PWD to Locked after three wrong passwords

State	Event	Next state
Enter ID	type ID, OK	Enter PWD
Enter PWD	password correct	Home
Enter PWD	wrong, tries < 3	Enter ID
Enter PWD	wrong 3 times	Locked

Benefits: simple, easy to understand and implement, supports checks for dead ends.
Limitations: state explosion for big dialogs, no concurrency or hierarchy (statecharts solve this).

Q2. Describe the Petri Net method and explain how it is used to design and evaluate user interfaces. Discuss the benefits and limitations of using Petri Nets in dialog design. (2024)

A **Petri net** is a graphical and mathematical model with **places** (circles, conditions), **transitions** (bars, events), **arcs** and **tokens** (dots). A transition is **enabled** when each input place holds a token; when it **fires** it removes a token from each input place and adds one to each output place.



Use in design: places represent conditions of the dialog (document ready, printer free), transitions represent user or system events (start, finish) and tokens represent the current state of the dialog and resources.

Example (print job): marking (Doc, Free, Printing, Done):

- $M_0 = (1, 1, 0, 0)$: document ready and printer free

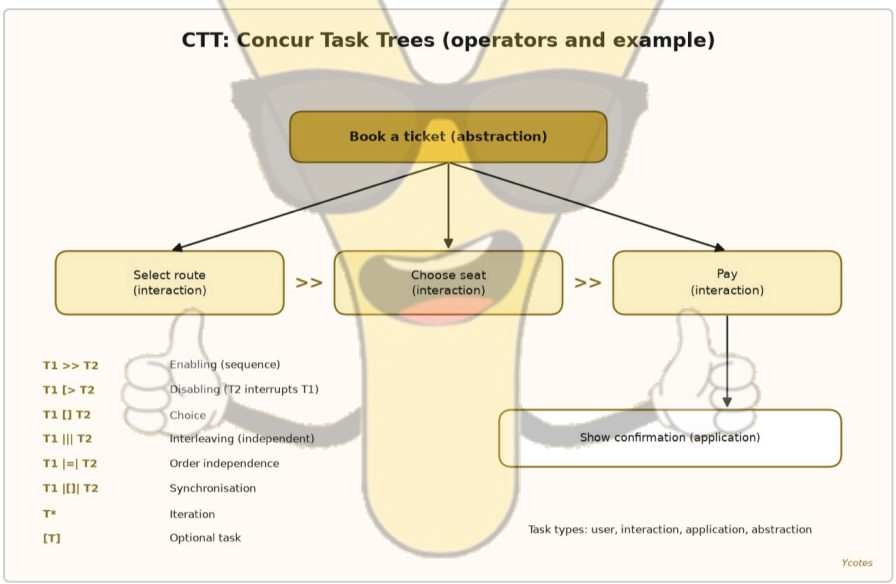
- **Start** fires: $M_1 = (0, 0, 1, 0)$
- **Finish** fires, returning the printer: $M_2 = (0, 1, 0, 1)$

Use in evaluation: analyse the reachable markings to find deadlock (no enabled transition), unreachable states, conflicts, and check concurrency and synchronisation of user actions.

Benefits: models concurrency, resource sharing and synchronisation naturally; formal analysis possible; graphical.
Limitations: nets become large and cluttered, no hierarchy in basic form, hard to show data and screen detail, and less intuitive for designers and users.

Q3. Explain Concur Task Tree. (Feb 2024)

CTT (Fabio Paterno) is a notation for task models that shows a task as a **tree** of sub-tasks connected by temporal operators, and that explicitly allows concurrency.



Task types:

- **User task:** done by the user alone (decide, think)
- **Interaction task:** the user acts on the system (click, type)
- **Application task:** the system performs it (show result)
- **Abstraction task:** a complex task made from sub-tasks

Temporal operators: enabling (>>), enabling with information passing, choice ([]), concurrency (|||), disabling ([>), suspend-resume (|>), order independence (|=|), iteration (*) and optional ([ ]).

Example: *Book a ticket* = Select route >> Choose seat >> Pay, and then the application task Show confirmation. If the user could pay by card or wallet, a choice operator links the two payment sub-tasks.

Advantages: hierarchical, expressive, supports concurrency, tool support (CTTE) for simulation and for generating user interfaces.
Limitations: large trees get complex and it needs training.

