10 Task Models

Task models are methods to analyze people's jobs, i.e. what they do, what things they work with and what they must know. There are several approaches to task analysis that we will study now in detail:

10.1 Task decomposition

In task decomposition, as it is easy to imagine, we split each task into ordered subtasks. The aims of this technique are to describe the actions people do, to structure them within task subtask hierarchy and describe order of subtasks. *Hierarchical Task Analysis (HTA)* is widely used as a technique for task decomposition.

Hierarchy description ...

0. in order to clean the house
    1. get the vacuum cleaner out
    2. get the appropriate attachment
    3. clean the rooms
        3.1. clean the hall
        3.2. clean the living rooms
        3.3. clean the bedrooms
    4. empty the dust bag
    5. put vacuum cleaner and attachments away

... and plans

Plan 0: do 1 - 2 - 3 - 5 in that order. when the dust bag gets full do 4

Plan 3: do any of 3.1, 3.2 or 3.3 in any order depending on which rooms need cleaning

N.B. only the plans denote order

Figure 53: HTA Description Example.

In order to generate the hierarchy, we do three steps:

1. get list of tasks
2. group tasks into higher level tasks
3. decompose lowest level tasks further

We expand only relevant tasks, hence we stop when subtasks are not needed anymore in a branch.

HTA diagrams are composed by explanations on nodes and plans about them:

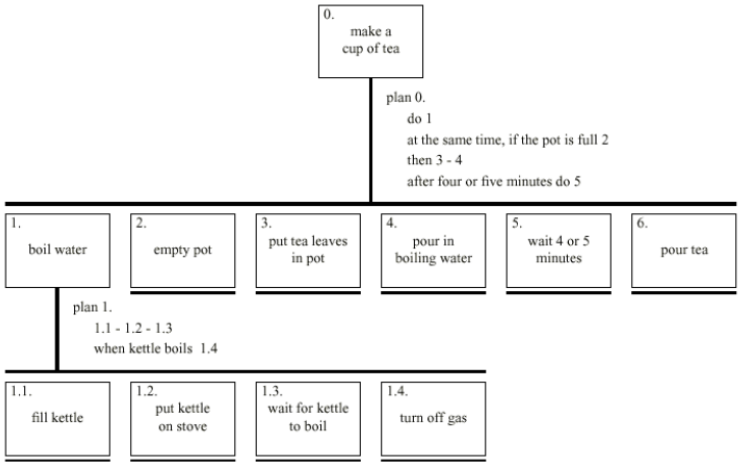


Figure 54: HTA Diagram Example.

We can also refine this diagram through heuristics, like restructure, balance and generalise:

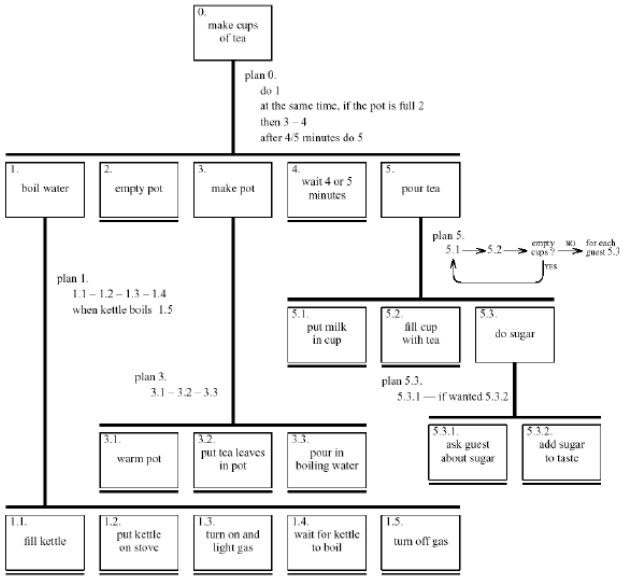


Figure 55: HTA Refined Example.

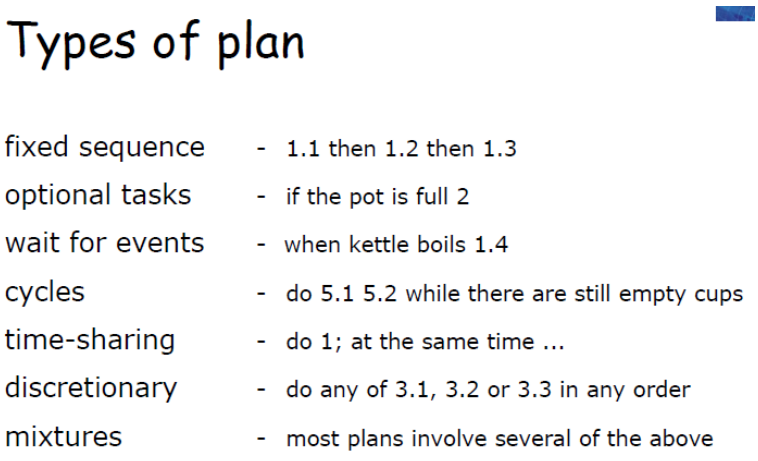


Figure 56: Plans.

Waiting can be misleading because we do not know where to put it, whether in tasks or plans.

10.2 Knowledge based analysis

In knowledge based analysis, the focus is on *objects* used in tasks and *actions* performed. *Taxonomies* represent levels of abstraction.

```
motor controls
   steering  steering wheel, indicators
   engine/speed
        direct   ignition, accelerator, foot brake
        gearing  clutch, gear stick
   lights
        external  headlights, hazard lights
        internal  courtesy light
   wash/wipe
        wipers   front wipers, rear wipers
        washers  front washers, rear washers
   heating  temperature control, air direction,
            fan, rear screen heater
   parking  hand brake, door lock
   radio  numerous!
```

Figure 57: Knowledge Based Example.

There are three types of branch point in taxonomy:

- XOR, normal taxonomy and object in one and only one branch
- AND, object must be in both, multiple classifications
- OR, weakest case can be in one, many or none

```
kitchen item AND
/____shape XOR
/    |____dished  mixing bowl, casserole, saucepan,
/    |            soup bowl, glass
/    |____flat    plate, chopping board, frying pan
/____function OR
     {____preparation    mixing bowl, plate, chopping board
     {____cooking    frying pan, casserole, saucepan
     {____dining XOR
          |____for food   plate, soup bowl, casserole
          |____for drink  glass
```

N.B. '/ | {' used for branch types.

Figure 58: Task Description Hierarchy Example.

10.3 Entity-Relationship Techniques

Here we have focus on objects, actions and their relationships.

Running example

'Vera's Veggies' – a market gardening firm
owner/manager: Vera Bradshaw
employees: Sam Gummage and Tony Peagreen
various tools including a tractor `Fergie`
two fields and a glasshouse
new computer controlled irrigation system

Figure 59: Entity-Relationship Technique Example.

We start with a list of objects and classify them:

- *Concrete objects*: simple things
- *Actors*: human actors
- *Composite objects*: sets and tuples

Attributes can be added to the objects. *Actions* instead are listed and associated with each agent (who performs the action), patient (which is changed by the action), instrument (used to perform action).

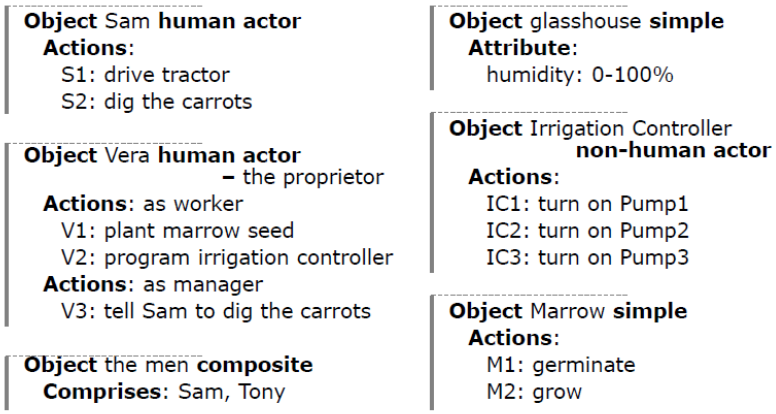


Figure 60: Object and Actions Example.

Events are performance of actions and can be spontaneous or timed. *Relationships* can be between objects, action-object, actions-events or temporal ones.

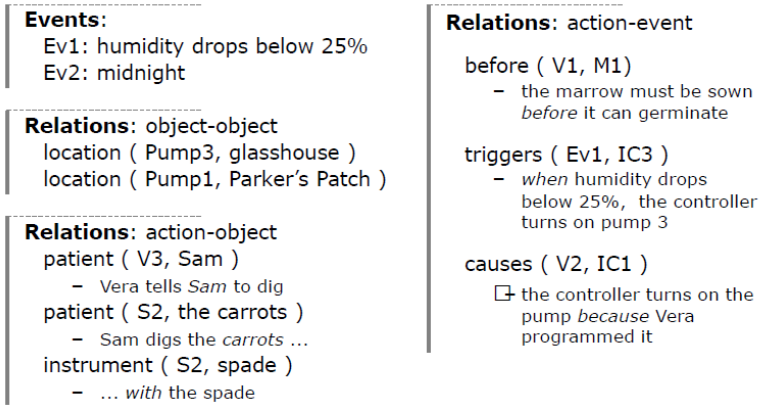


Figure 61: Events and Relationships Example.

Sources of Information can be documentation, observations and interviews.

11 Dialogue Notations and Design

A dialogue is a conversation between two or more parties, usually cooperative. In UIs, it refers to the structure of the interaction, syntactic level of human-computer conversation. There are several levels, such as *lexical* (shape of icons, actual keys pressed), *syntactic* (order of inputs and outputs) and *semantic* (effect of internal application/data). Note that human-human dialogue is usually very unstructured, while in computers we always have a fixed and constrained structure. Dialogue gets buried in the program; in a big system we can analyze the dialogue because dialogue notations help us to analyze systems and separate lexical from semantic. Before the system is built, notations help us understand proposed designs.

11.1 Graphical Notations

11.1.1 STNs

STNs, i.e. State Transition Networks, are diagrams characterized by states (circles) and arcs (actions/events) and describe the behavior of a system in a particular path/function.

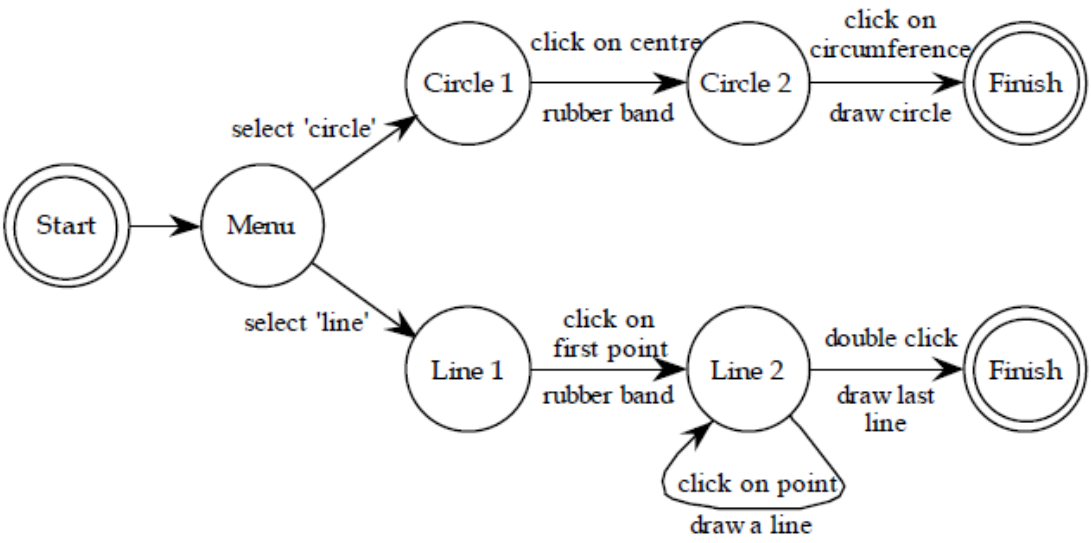


Figure 62: STN Example.

As we can see from the figure, arc labels are a bit cramped because notation is "state heavy" and the events require most detail. Labels in circles are a bit uninformative instead: in fact, states are hard to name but easier to visualize. STNs can also be organized in a hierarchy, such that they can manage complex dialogues. Concurrent dialogues instead, are like what happens in a concurrent dialog checkbox:

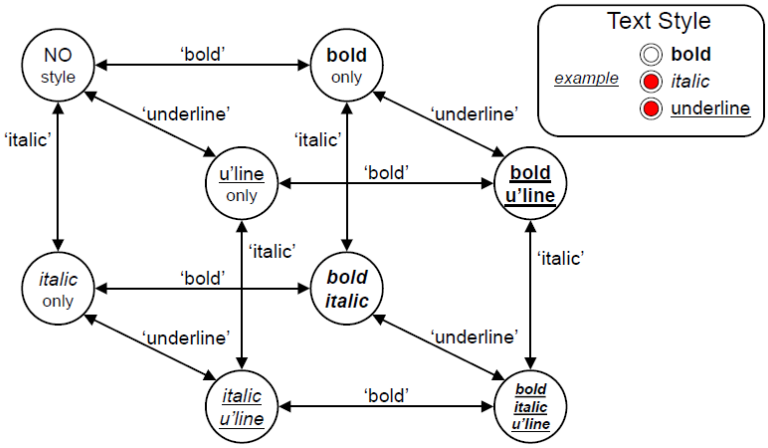


Figure 63: Concurrent Dialogues Example.

Escapes are actions which permit the user to exit from an STN-related app function, like back button or similar. We have to avoid this, generally, and provide an easy-to-use escape to the user instead of a trivial back button.

11.1.2 Petri nets

Petri nets are one of the oldest notations in computing. They are flow graphs, with *places*, *transitions* and *counters*. First two ones are a bit like in STNs, while a counter is the current state. Several counters are allowed, i.e. concurrent dialogue states.

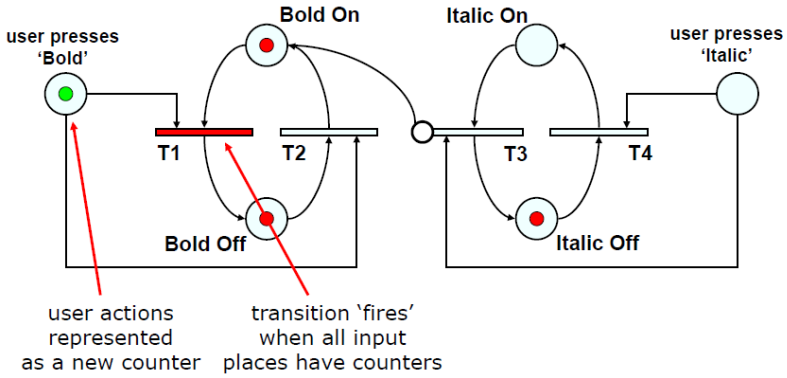


Figure 64: Petri Net Example.

11.1.3 State charts

State charts are used in UML and are an extension to STNs, with hierarchy, concurrent subnets, escapes and history.

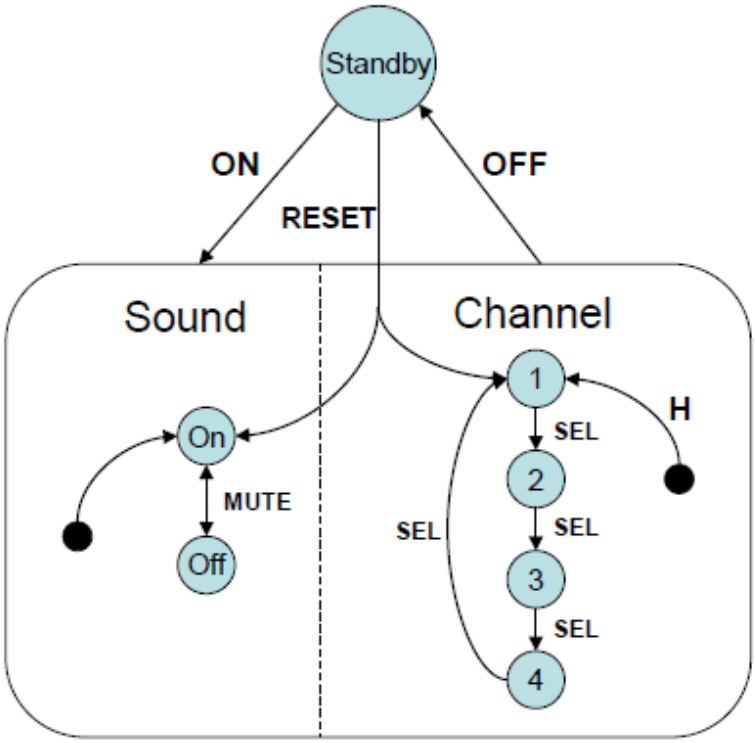


Figure 65: State Chart Example.

11.1.4 Flowcharts

Flowcharts are familiar to programmers and are composed by process/event, not states. They are used for dialogues. There is a COBOL transaction processing and, as we can see frm the following figure, dialogue flow charts are used.

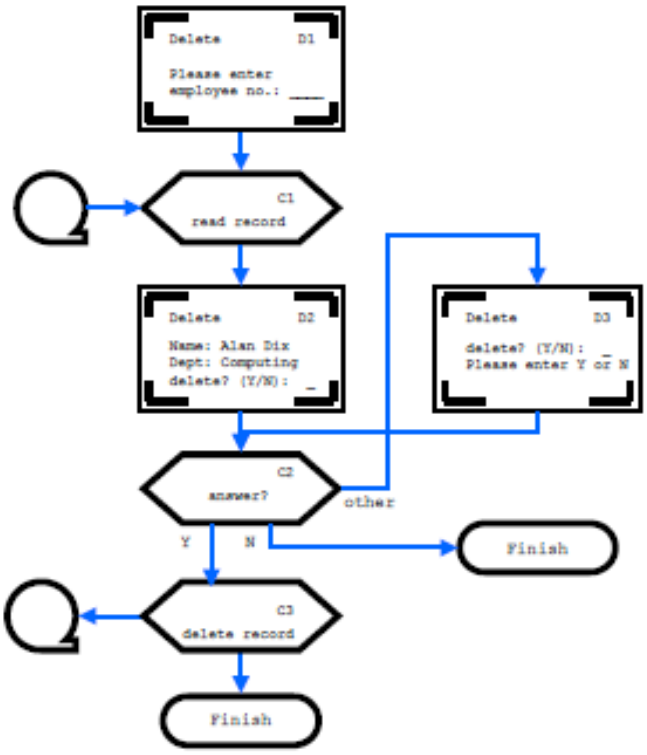


Figure 66: Flow Chart Example.

11.1.5 JSD Diagrams

They are used for tree structured dialogues. They are less expressive but have greater clarity.

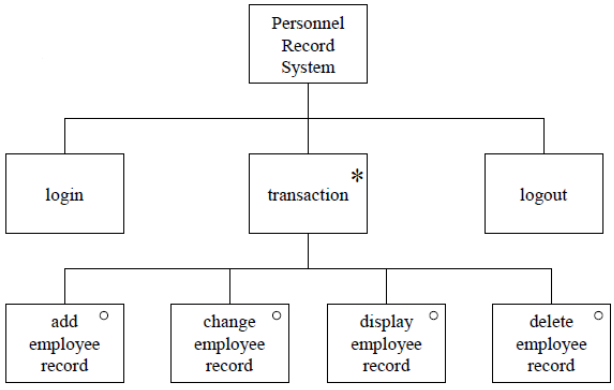


Figure 67: JSD Example.

11.2 Textual Notations

11.2.1 Grammars

- Regular expressions
  ```
  sel-line click click* dble-click
  ```
- compare with JSD
  - same computational model
  - different notation
- BNF
  ```
  expr ::= empty
         | atom expr
         | '(' expr ')' expr
  ```
- more powerful than regular exp. or STNs
- Still NO concurrent dialogue

11.2.2 Production rules

It is an unordered list of rules of the type "if condition then action". They are good for concurrency but bad for sequence!
Event-based production rules are rules added to list of pending events, as they represent events themselves.

```
Sel-line → first
C-point first → rest
C-point rest → rest
D-point rest → < draw line >
```

Figure 68: Event-based prod. rule example.

Prepositional Production System is state based and has attributes and rules.

Attributes:

Mouse: { mouse-off, select-line, click-point, double-click }
Line-state: { menu, first, rest }

Rules (feedback not shown):

select-line → mouse-off first
click-point first → mouse-off rest
click-point rest → mouse-off
double-click rest → mouse-off menu

Figure 69: Attributes and rules example.

11.2.3 CSP and Process Algebras

They are used in Alexander's SPI that we will now analyze; they are good for sequential dialogues and concurrent dialogue, but causality is unclear in them.
In *Alexander's SPI*, we have two-parts specification (*EventCSP* pure dialogue order and *EventISL* target dependent semantics). Dialogue description is centralised and syntactic/semantic tradeoff tollerable.

Semantics Alexander SPI (ii)

- EventCSP

```
Login  = login-mess -> get-name -> Passwd
Passwd = passwd-mess -> (invalid -> Login [] valid -> Session)
```

- EventISL

```
event: login-mess
    prompt: true
    out: "Login:"
event: get-name
    uses: input
    set: user-id = input
event: valid
    uses: input, user-id, passwd-db
    wgen:  passwd-id = passwd-db(user-id)
```

Here we describe *Action Properties*:

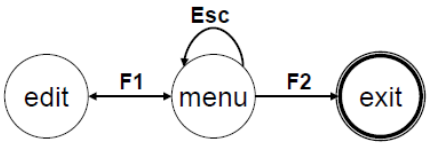
- *completeness*
- *determinism*
- *nested escapes*
- *consistency*

these properties can be checked in an STN: in fact, a good UI designer should take care of them a lot.
State Properties are:

- *reachability*
- *reversibility*
- *dangerous states*

Dangerous States

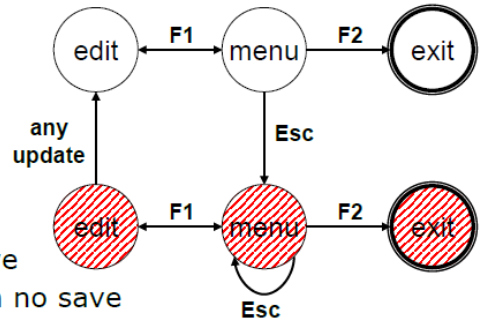
- word processor: two modes and exit
  - F1 - changes mode
  - F2 - exit (and save)
  - Esc - no mode change



but ... Esc resets autosave

Dangerous States (ii)

- exit with/without save ⇒ dangerous states
- duplicate states - semantic distinction



F1-F2 - exit with save
F1-Esc-F2 - exit with no save

Note that such dangerous states can be reached for fault of layout, so we have to design a layout that avoids errors. Think, for example, that we have two nearby buttons for opposite actions. If the user wrongly clicks the other one, it will trigger the opposite action and it could be a disaster in very important, not reversible actions!