Icons design

Icons design

- **Icon ...small visual symbol (general definition)**
  - **``A picture is worth a thousand words."**
- ***Well-designed* icons:**
  - **Save space on the screen**
  - **Easily and quickly recognized, even in full visual environment**
  - **Easily memorized**
  - **Help to make international interfaces**

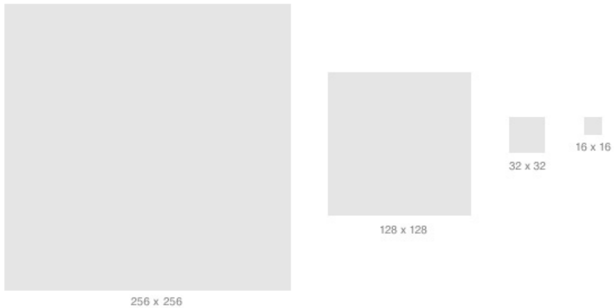
Icons design

- What companies these logos represent?

Icons design

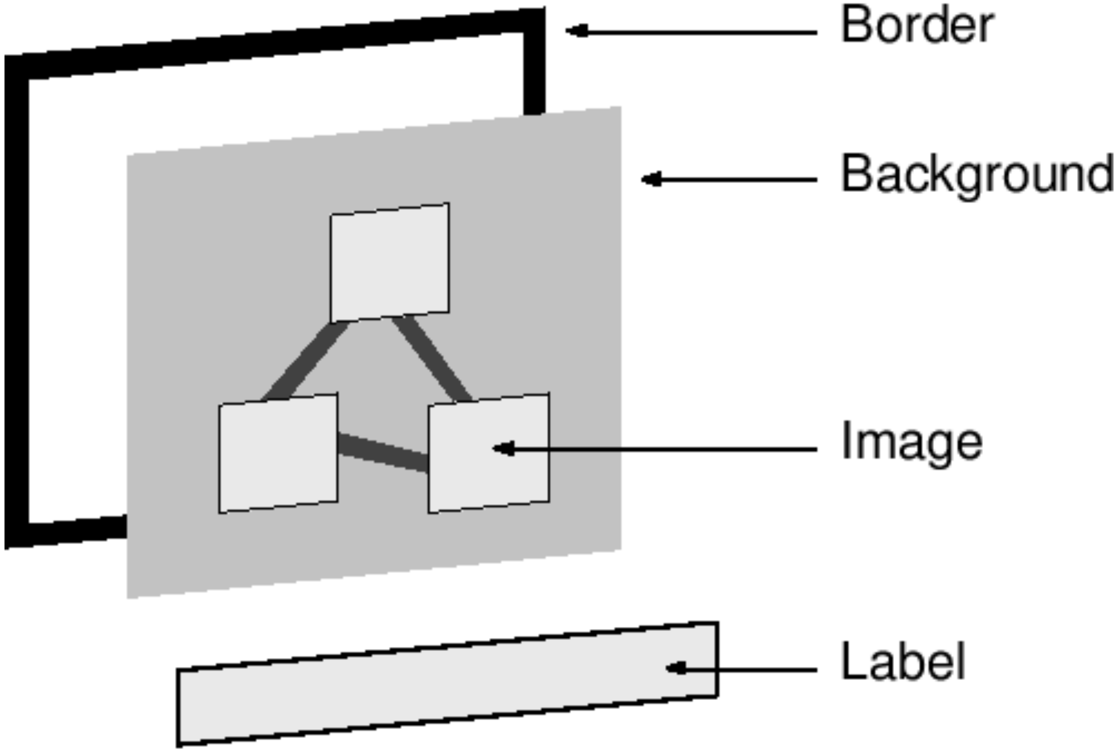
- Icons are not necessarily scalable

- Icons are restricted to a quadratic area



Icons design

- **Standard components of a icon:**
  - **Border**
  - **Background**
  - **Image**
  - **Label**

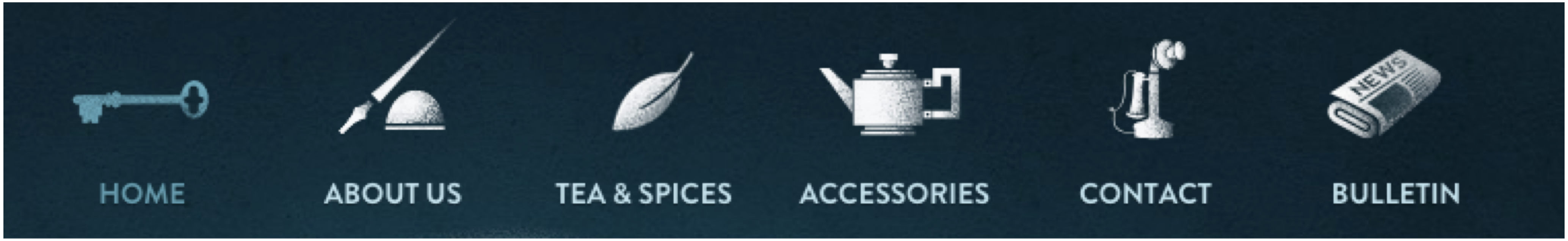


Icons design

- Design principles

  1) Coherency

  - **Use one icon set style**



  - **Consistency in terms of colour, lighting, perspective, metaphor, level of realism (abstraction).**
  - **In a group, icons should be visually balanced.**
  - **Visual distinctions should have a meaning – too much decoration is distracting.**

Icons design

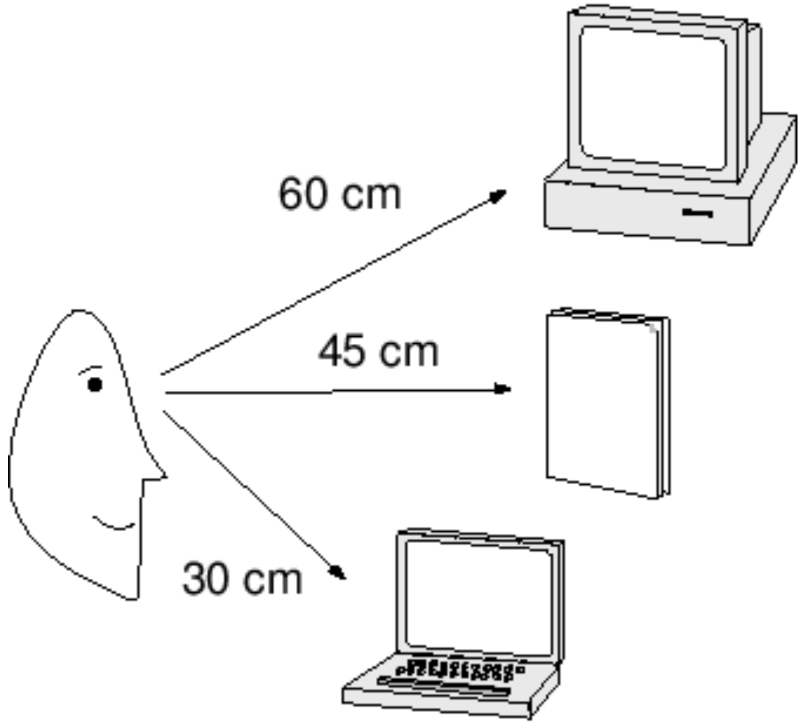
- Design principles

  2) Legibility

  - Use big objects, bold lines and simple areas.
  - Consider the display size and resolution and the distance from the display to the user.
  - Good contrast “foreground/background”.
  - External shape (silhouette) reveal information.

Icons design

- Design principles
  2) Legibility
  - Typical visualization distances:
    - Desktop monitor: 60cm.
    - Papel document: 45cm.
    - Laptop screen: 30cm.



Icons design

- Design principles

  3) Recognition and recall

  - Whenever possible, choose a familiar metaphor.
  - Use concrete objects. Concepts and abstract actions are difficult to visualize.
    [Do you know a good icon for ``Undo''?]
  - Provide textual labels; tooltips.

  4) Save colours

  - First design in B/W, add colour later.

Icons design

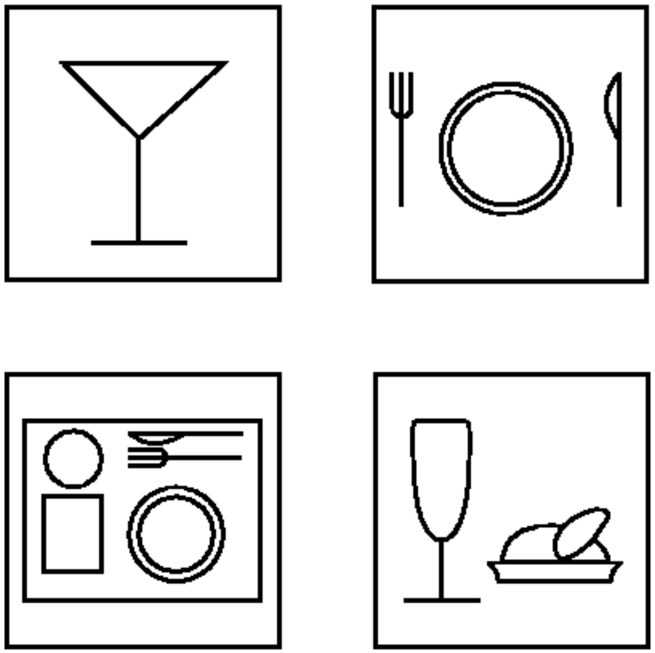
Icons design

Icons design

- Cultural issues
  - Avoid to include text or alphabetic characters in icons. Use labels or tooltips to avoid changes in the icon when the interface has to be translated into another language.
  - Facial expressions can vary across cultures – do not use it in icons.
  - Metaphors may depend on cultures.

Icons design

- Icons are not always appropriate
  - For more abstract concepts and subtle distinctions, verbal representations can sometimes work better than iconic representations.



Bar

Snacks

Selfservice

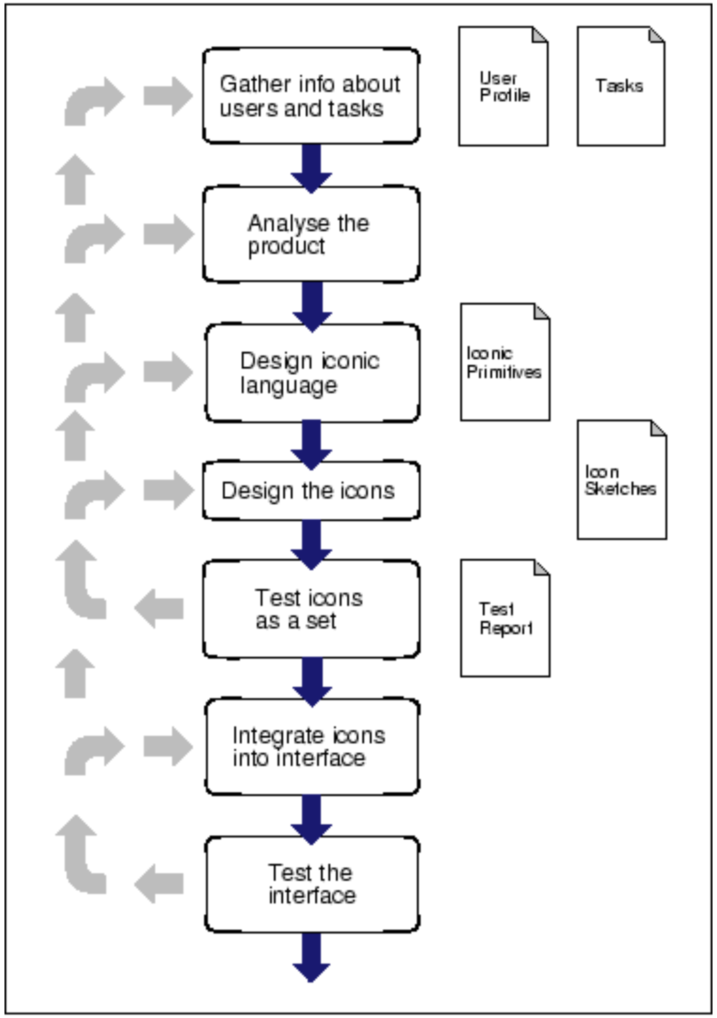
Restaurant

Icons design

- Iconic language
  - For large sets of icons, it becomes convenient to develop an iconic language.
  - An iconic language is a systematic way of combining elementary symbols to create more complex icons:
    - *Vocabulary*: set of primitive symbols.
    - *Grammar*: combination rules.

Icons design

- Life cycle



Icons design

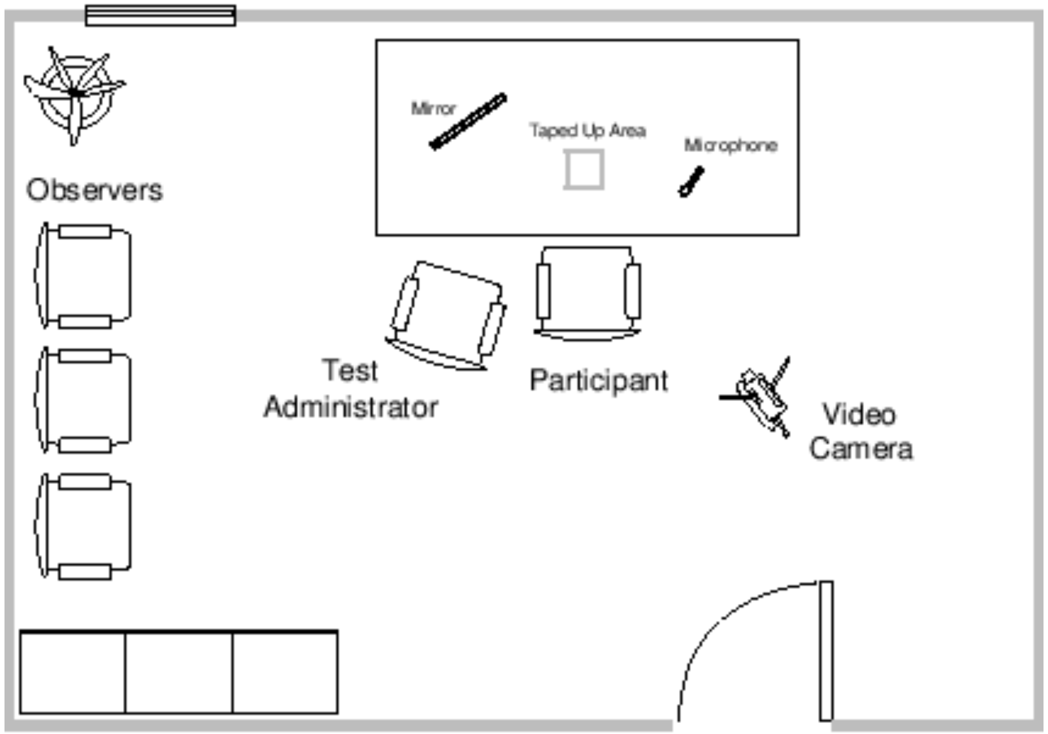
- Life cycle
  - Iterative design
    - Start with a simple B/W icon, create paper sketches.
    - Test and redesign until you find basic symbols.
    - Add grey shades or a few colours. Design in the computer. Print colour versions in real size.
    - Test and redesign until you achieve the desired result.

Icons design

- Life cycle
  - Icon (or set of icons) intuitiveness test:
    - The icon (or icons) is shown (without label) to some users (typically 5). The users should say what they think the icon represent.
    - After the test, interview the users to collect more detailed data (using icon prints).
    - This test allows to evaluate how well an icon represent the desired concept.
  - Usability – think aloud:
    - The icons should be placed in order to represent the interface as a whole.
    - Ask the user to think aloud while using the interface. Capture the user initial reaction and what he/she things the icons represent.
    - This test allows to evaluate how a certain icon behaves in the context of the whole interface.

Icons design

- Life cycle
  - Intuitiveness test



Dialog design

Dialog

- Conversation between two or more participants.
- Syntactic level of HCI
- Design of user interfaces: Structure of the conversation between the user and the system.
- 3 levels of computer languages:
  - Lexical: Lowest level: icons' shape, and keys pressed (words' sound and spelling).
  - Syntactic: Order and structure of the inputs and outputs (grammar and sentence construction).
  - Semantic: Meaning of the conversation in terms of its effect on the computer's internal data structures and/or the external world (meaning attributed by the different participants to the conversation).

Dialog

- In user interfaces, dialog is often connected to the syntactic level.

- Dialog is also connected with:
  - Semantic of the system – What it does?
  - System presentation – appearance

- Structured and with constraints

- Notations to describe human-computer dialogs:
  - Facilitates the analysis and the separation between the interface elements and the program calculation.
  - Allows the elaboration of the dialog structure before coding.

Dialog notations

- diagrammatic: easy to read at a glance
- textual: easier for formal analysis

Dialog notations

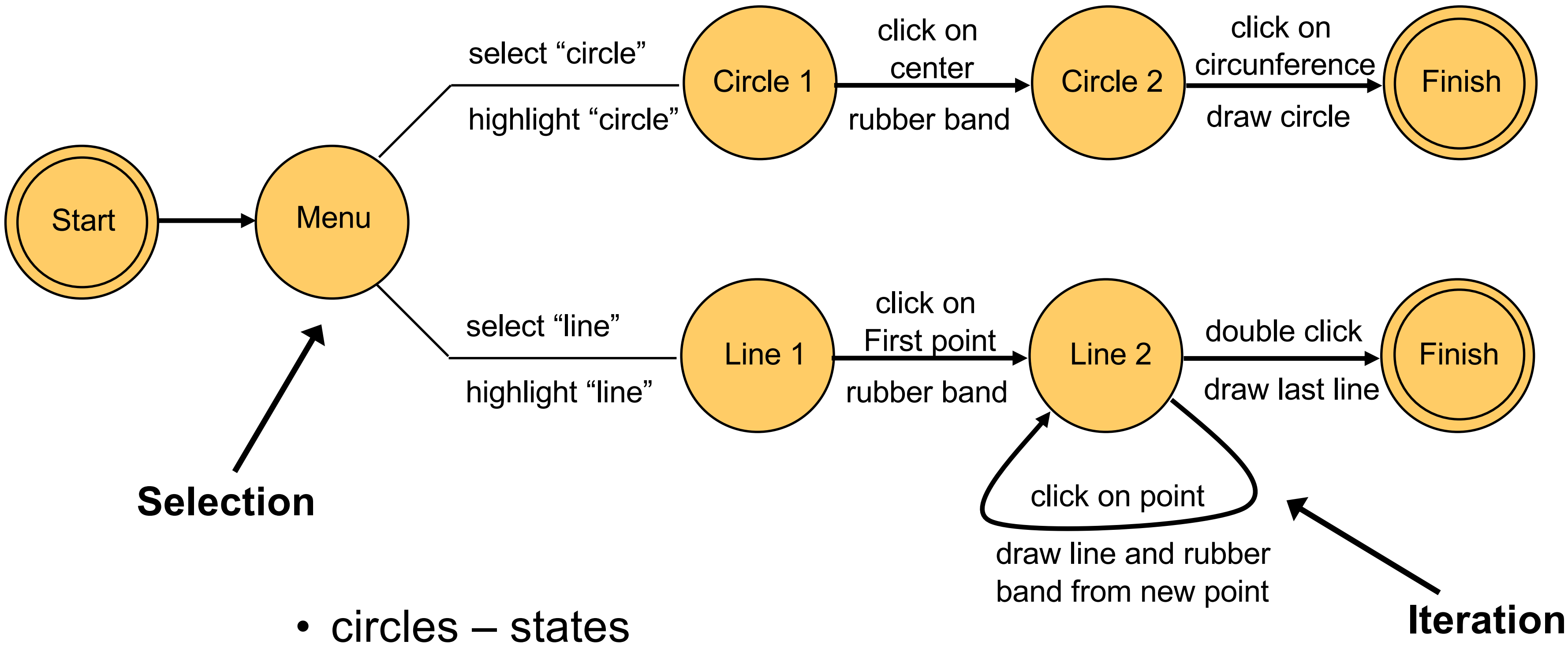
- Diagrammatic notations:
  - State Transition Networks (STN)
  - Petri nets
  - State diagrams
  - Flow charts
  - JSD (Jackson Structured Design) diagrams

State Transition Networks

- Components
  - States: set of attribute values that characterize the system (in one moment). Define a system visible behaviour that last for a period of time.
  - Transitions: Transitions are instantaneous changes in state (system dynamics).
  - Actions/responses: actions executed by the user, causing a state transition/response by the system.

State transition network

- Example: Drawing tool



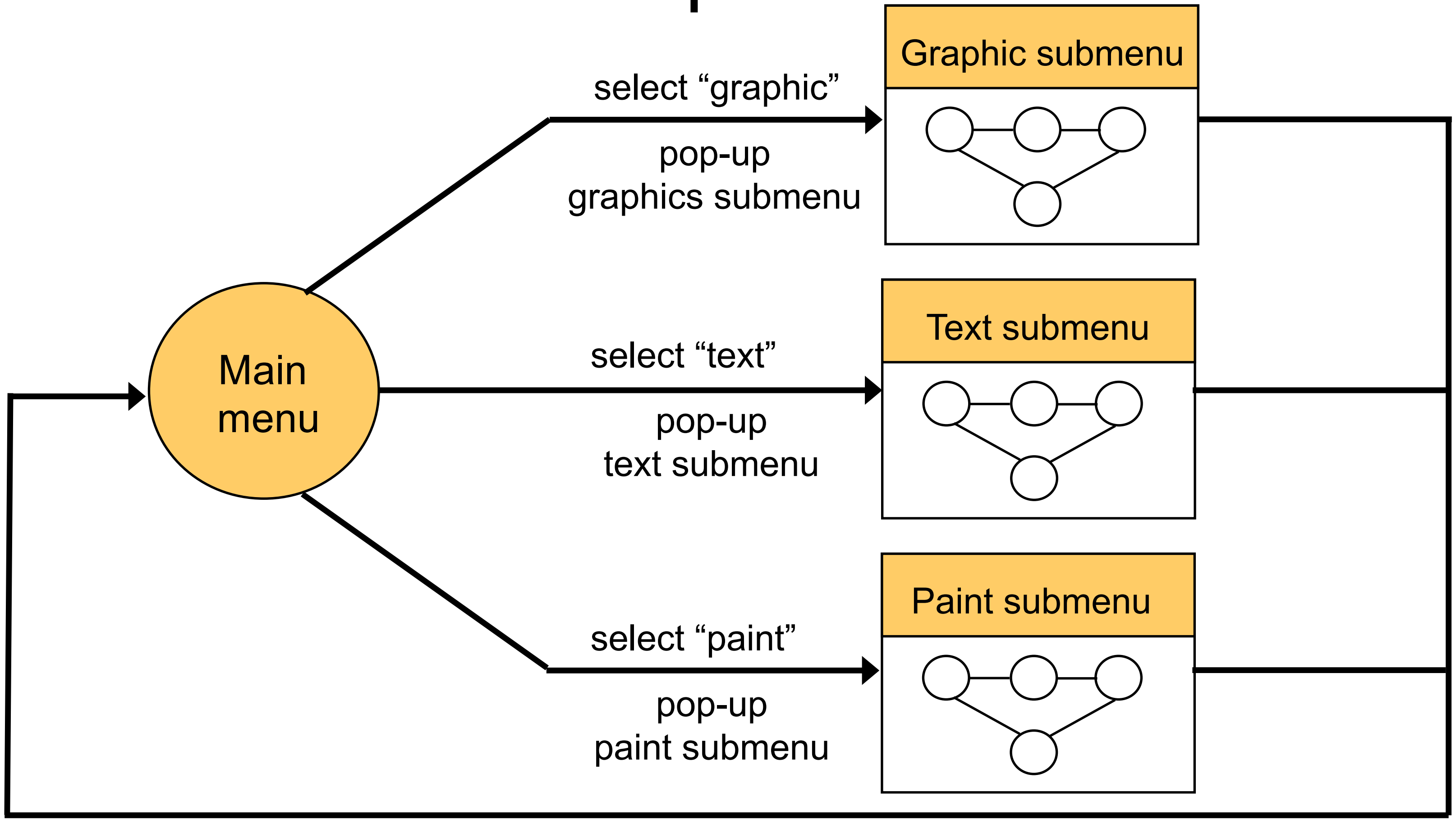
- circles – states
- arcs – actions/events

State Transition Networks

- Hierarchical decomposition
  - Simplifies the representation of complex networks.
  - Helps to build prototypes
    - Paper:
      - State → screen (hand draw or printed)
      - Follow the STN and present the screens according to the user actions.
    - Computer:
      - Multimedia authoring tool
      - State → screen
      - Add buttons and links to the desired screen (according to STN).

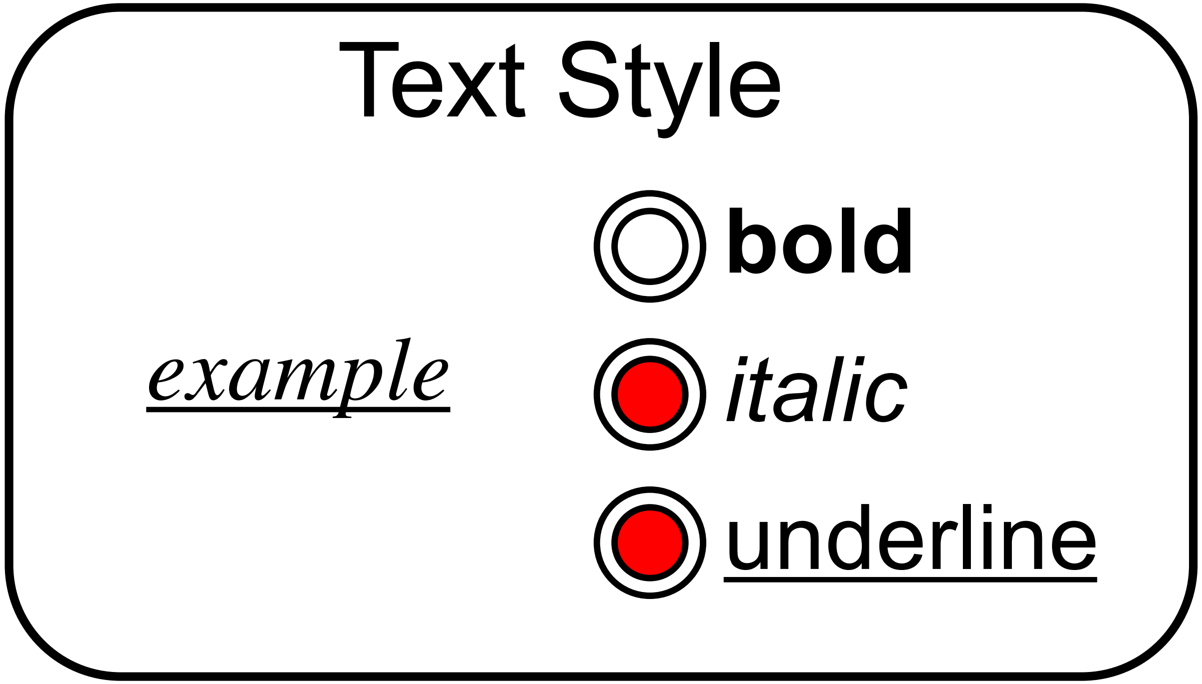
State Transition Networks

- Hierarchical decomposition



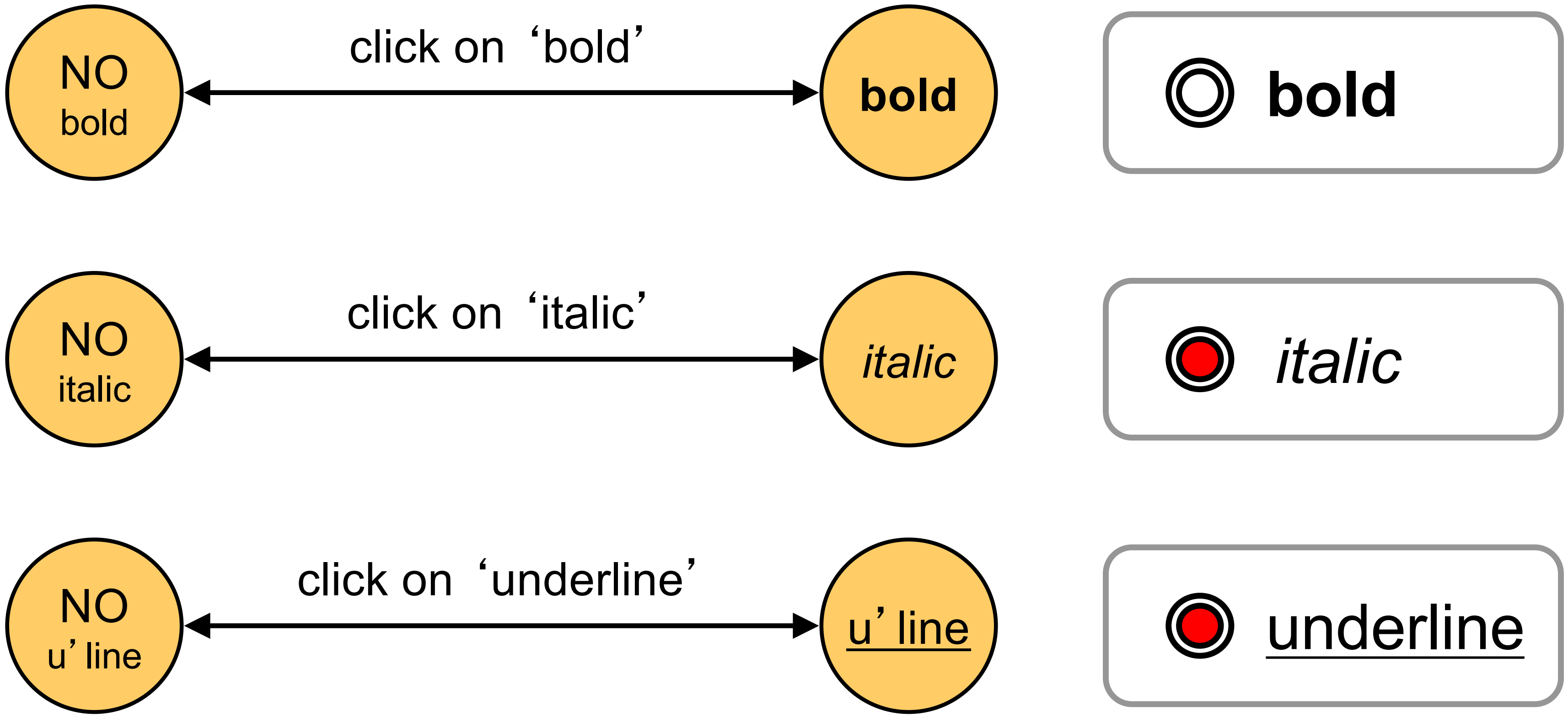
State Transition Networks

- Concurrent dialogs



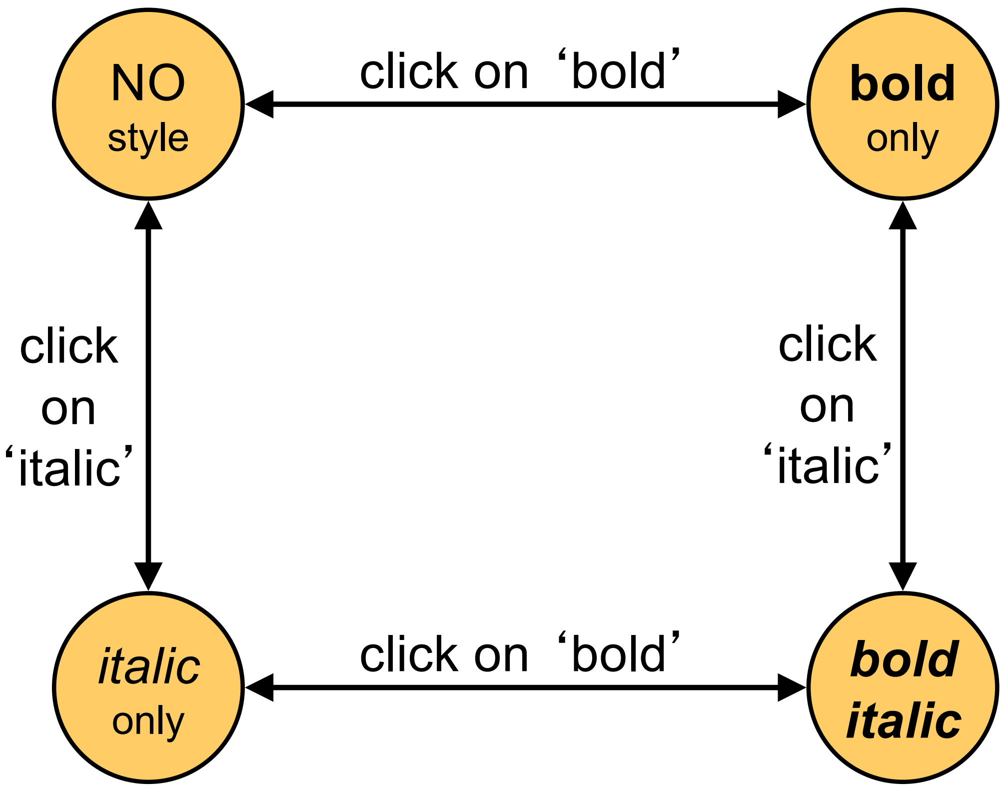
State Transition Networks

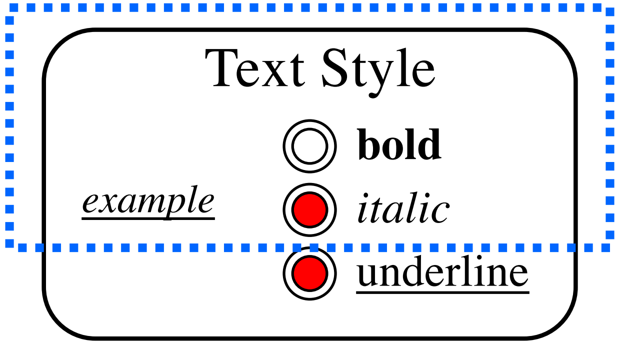
- Concurrent dialogs



State Transition Networks

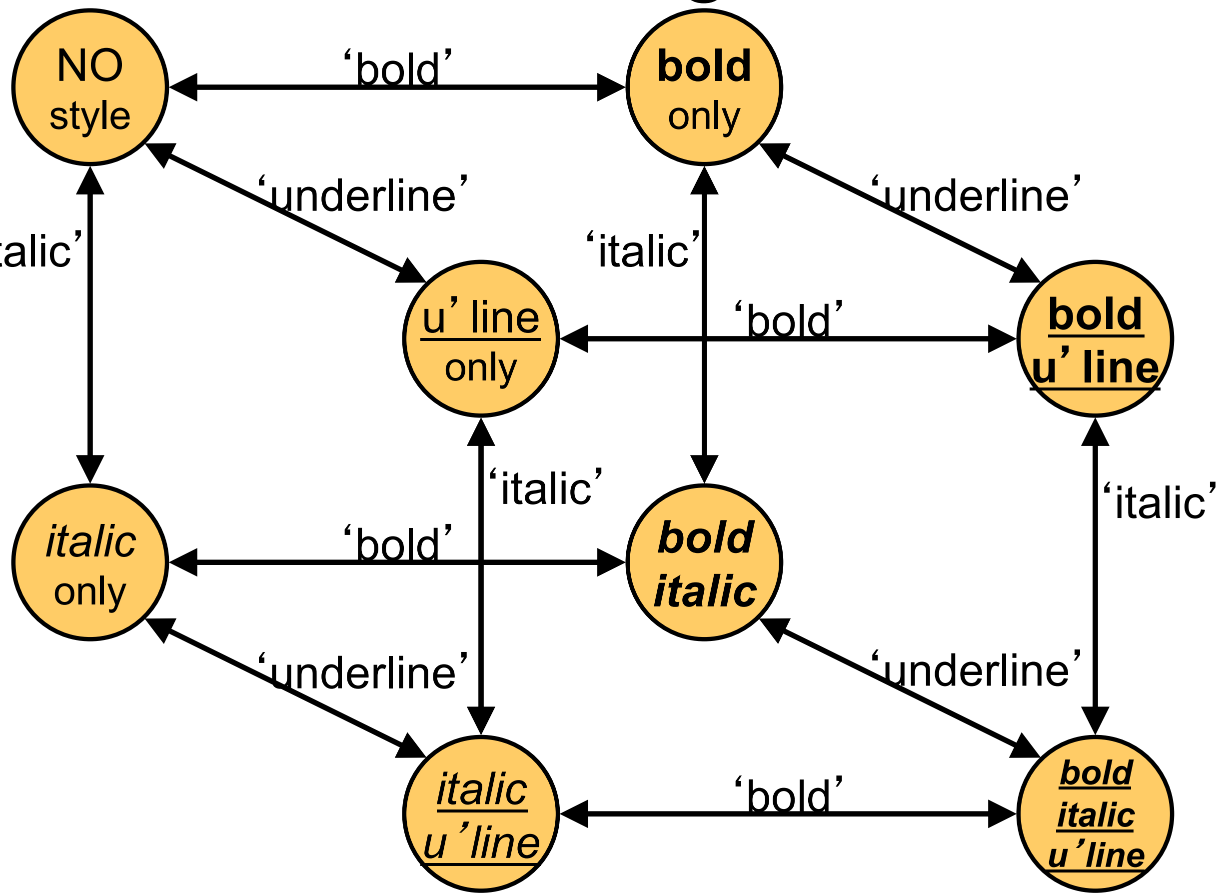
- Concurrent dialogs

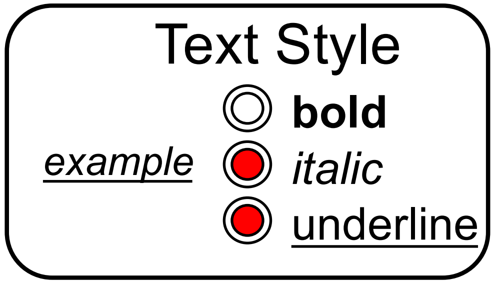




State Transition Networks

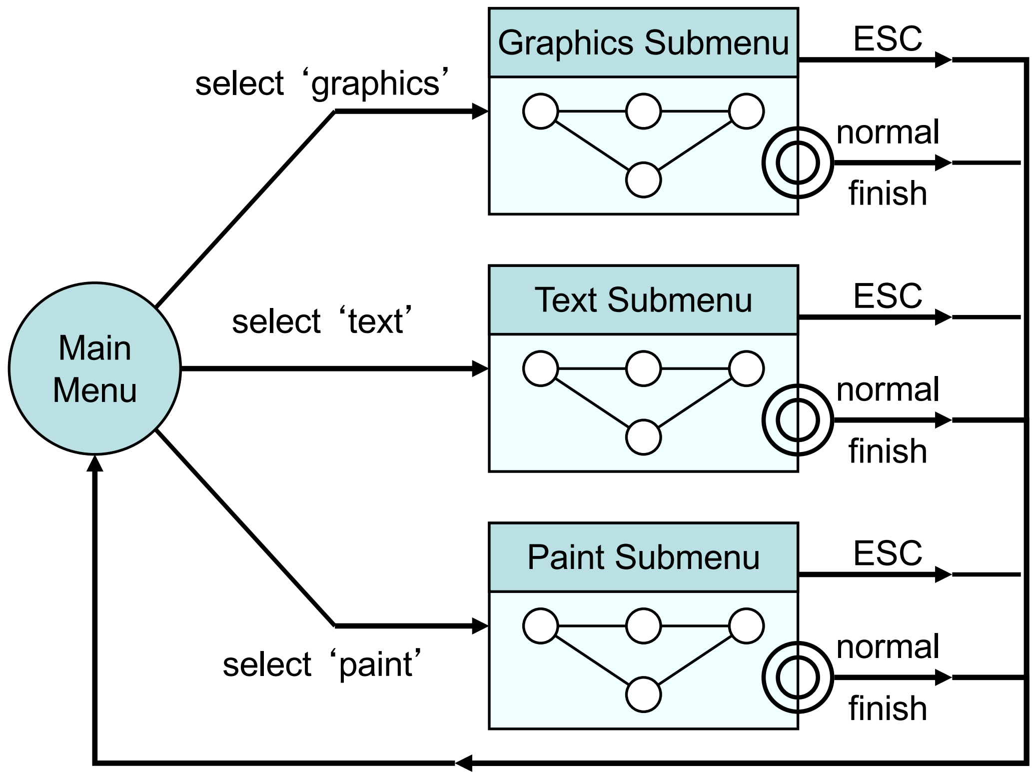
- Concurrent dialogs





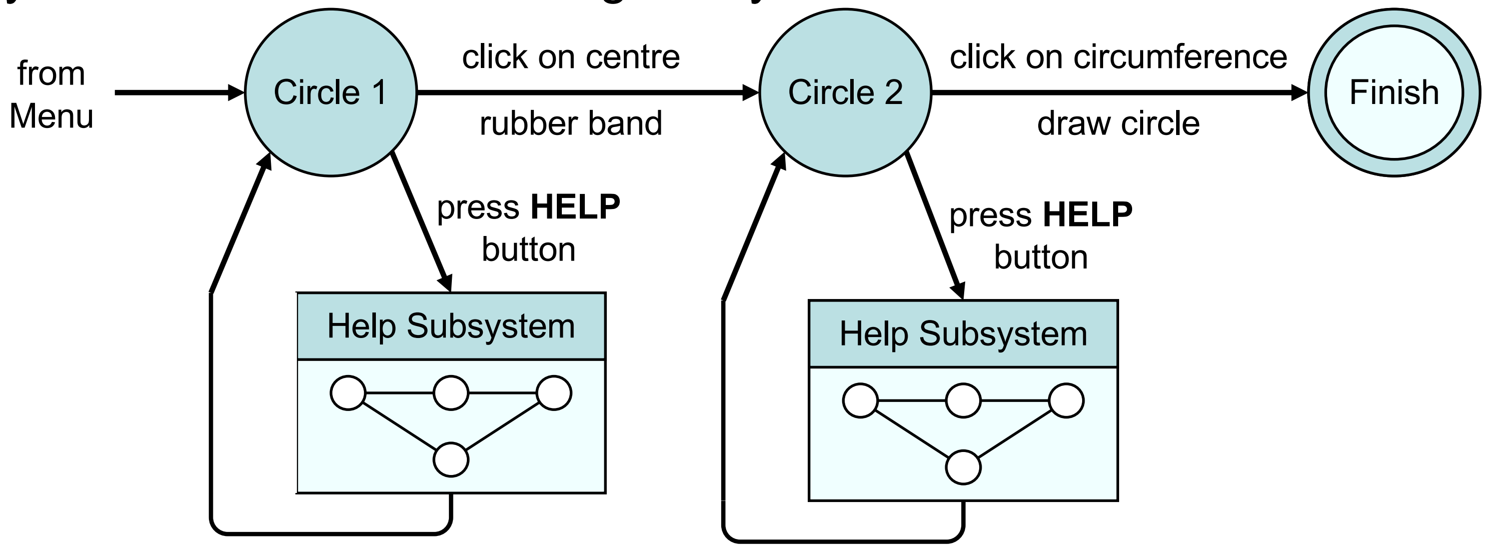
State Transition Networks

- Escapes
  - Be able to return from any state to the menu → 1 arc from every state back to the main menu.
  - Each submenu has 2 output arcs:
    - Normal – from “finish” state
    - ESC – active during the whole sub-dialog execution.



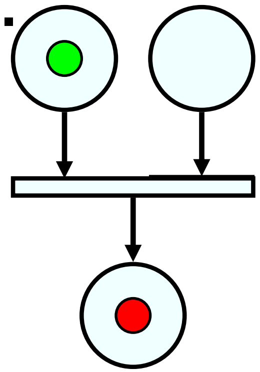
State Transition Networks

- Help
  - Can be invoked from any state during the dialog.
  - Returns to the same point in the dialog that you left.
  - Sub-dialog hanging off every state in the network.
  - Diagram becomes confusing → nº of states = nº of states in the help system * nº states in the original system.



Petri networks

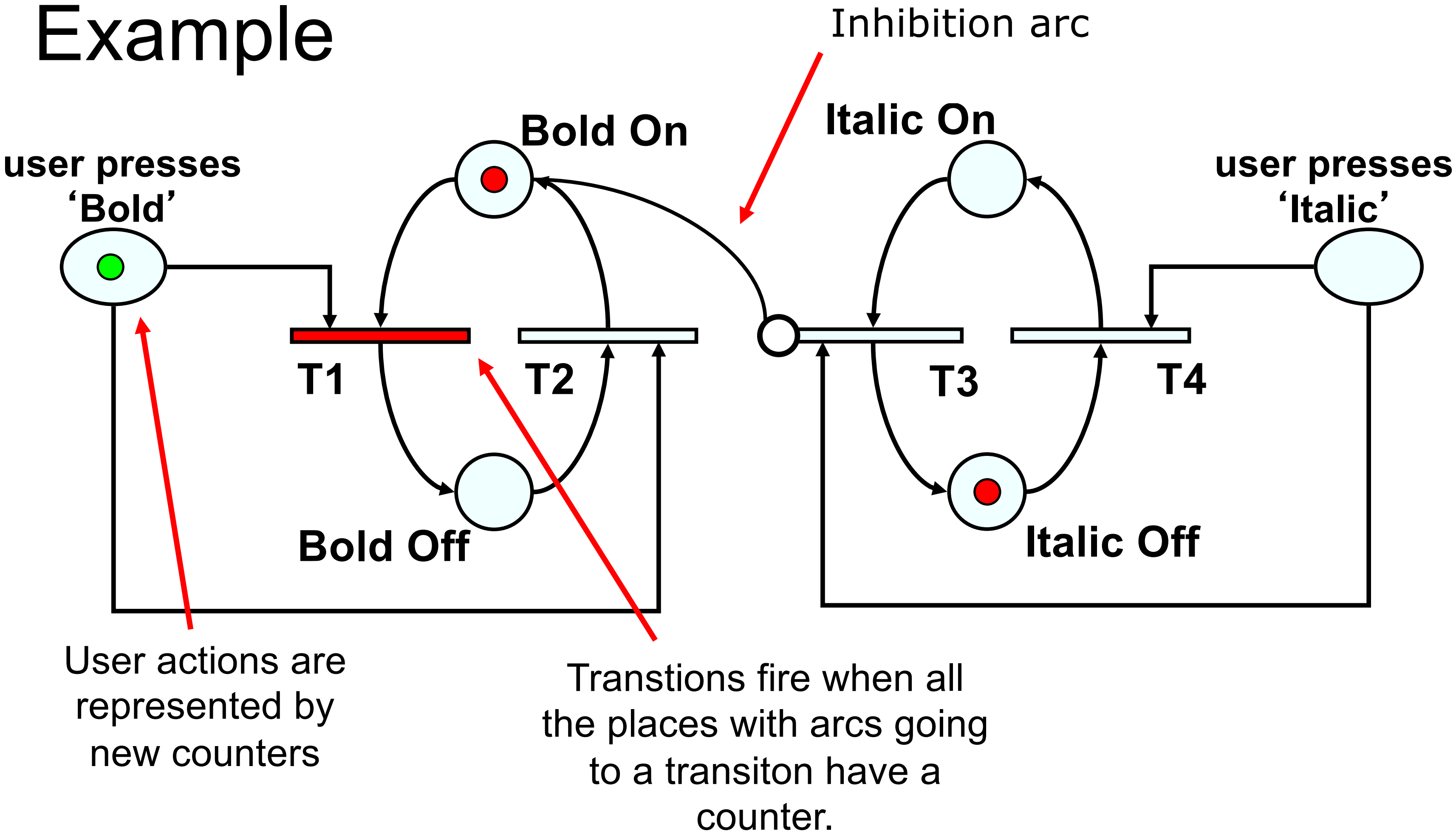
- One of the oldest formalisms in computing sciences.
- Graphical formalism designed for reasoning about concurrent activities.
- The system has several “states” at once.
- Graphical representation:
  - Places (states in STN)
  - Transitions (arcs in STN)
  - Counters (sign the current state)



- There can be several counters at the same time.
  - concurrent dialogs

Petri networks

- Example



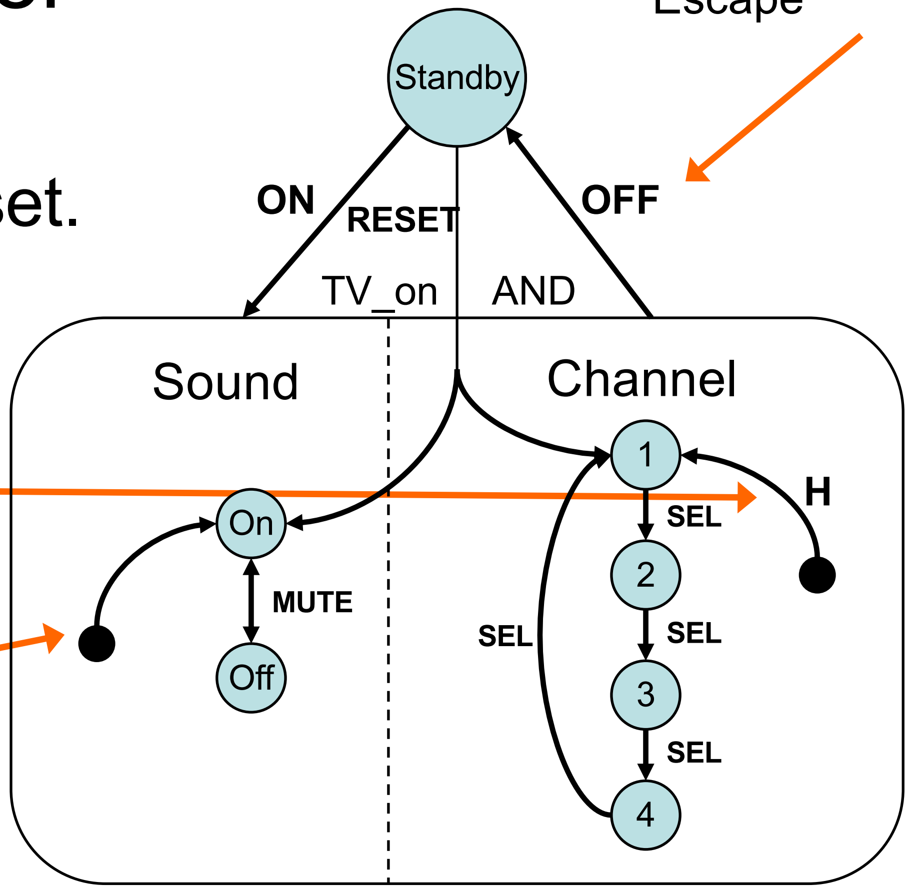
State diagrams

- Used in UML
- Visual specification of complex reactive systems.
- STN extension:
  - hierarchy
  - concurrent activity
  - escapes
  - history

State diagrams

- Example: TV controller
  - 5 buttons:
    - on, off, mute, sel e reset.

Works as
Escape

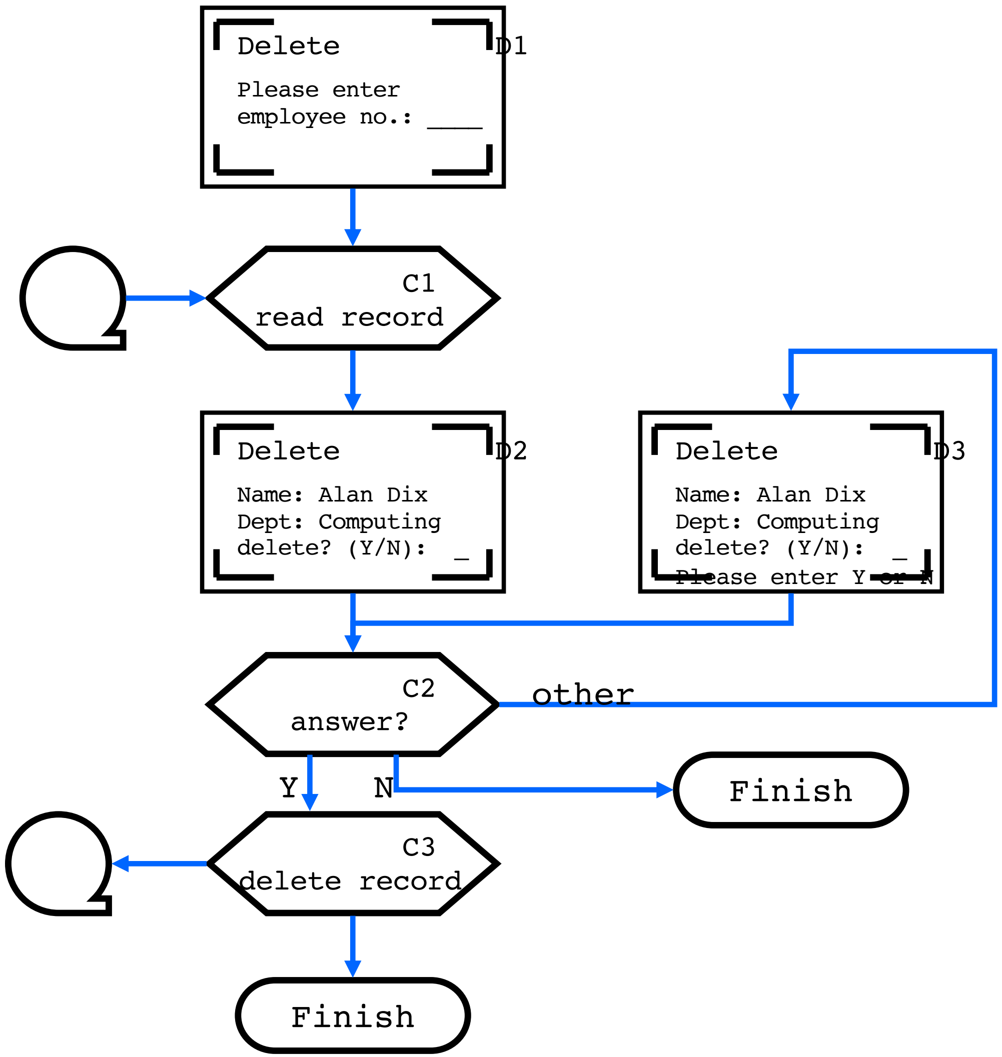


"history":
-1st time (or after "reset") starts on 1
- goes to former selected channel.

"start node" in the STN – represents the "default"

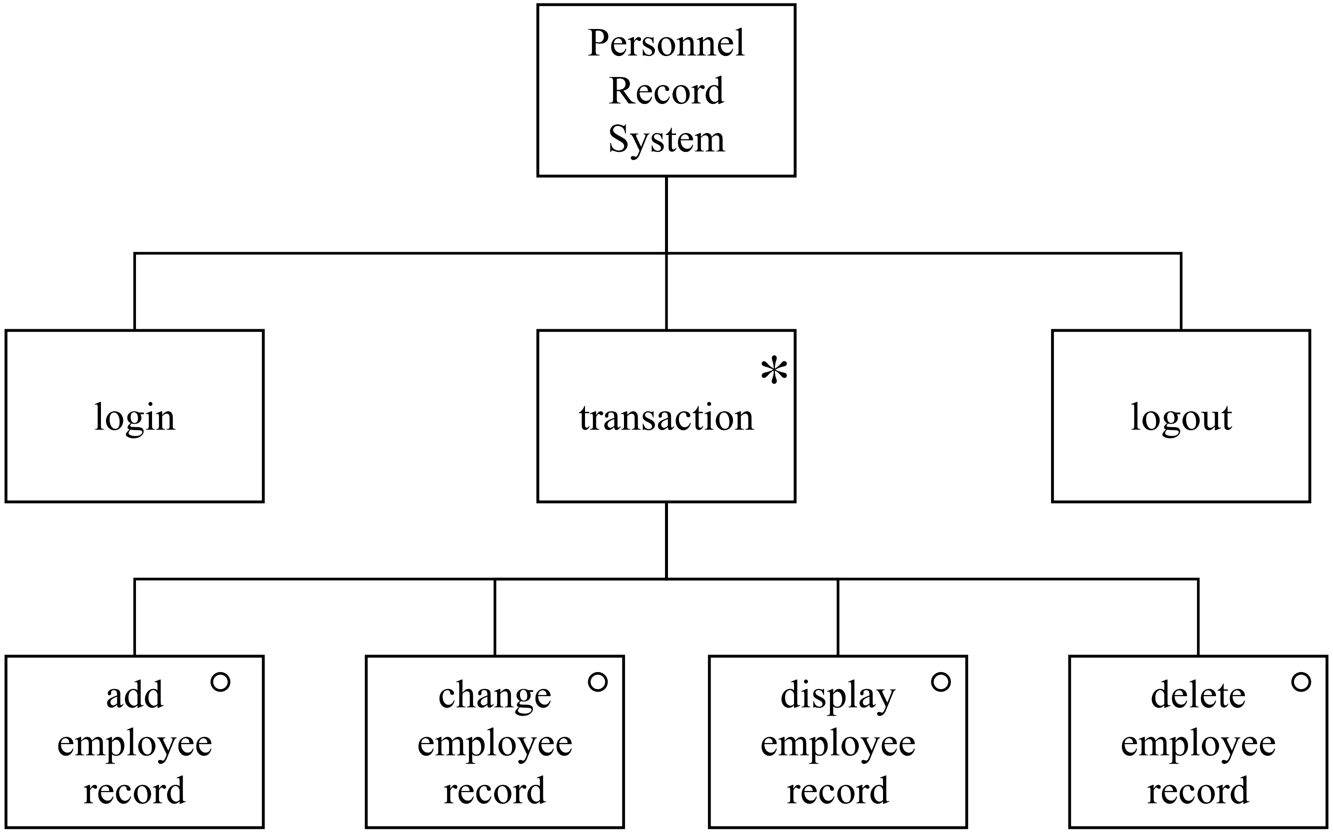
Flow charts

- Familiar to programmers
- Simplicity
- Similar problems to STN:
  - Concurrent activity
  - Escapes
  - ...
- Boxes:
  - processes/events
  - not states
- Represent dialog
  - Not the internal algorithm



Jackson Structured Design diagrams

- Simple
- Clear
- Limited
- Application:
  - High-level dialog specification
  - Menu based interfaces
  - Tree structured dialogs



Textual notation

- Grammars
  - BNF (Backus-Naur Form)
    - Dialog syntactic level
    - Used widely to specify the syntax of computer programming languages.
  - Example: line-drawing function

sequence

draw-line	::=	*select-line + choose-point + last-point*
select-line	::=	*position-mouse + CLICK-MOUSE*
choose-point	::=	*choose-one*
		| choose-one + choose-point
choose-one	::=	*position-mouse + CLICK-MOUSE*
Last-point	::=	*position-mouse + DOUBLE-CLICK-MOUSE*
position-mouse	::=	*empty | MOVE-MOUSE + position-mouse*

choice

recursive definition

Textual notation

- Grammars
  - Regular expressions
    - Programming languages lexical analysis.
    - Many different types with different notations.
    - Focus on the user's actions.
  - Example – Polyline drawing
    - Select-line click click* double-click
  - Similar to BNF (Backus-Naur Form), but less powerful.
  - Do not allows to represent concurrent dialogs and escapes.
  - Low-level dialogs (description of individual widgets).

Textual notation

- Production rules
  - List of rules with no implicit order
  - General form:

If *condition* then *action*

    - Conditions based on the state and pending events
    - Rules are always active
    - System constantly matches the condition part of the rules against the user-initiated events which have occurred. When the condition of the rule becomes true, the rule fires and the action part is executed (causing a response from the system or a change in the system memory).
  - Good for concurrency
  - Bad for sequence

Textual notation

- CSP (Communicating Sequential Processes)
  – Appropriate to represent concurrent and sequential actions.

Events (lower case)
Processes (upper case)

? – user actions
... – internal events
= – definition
→ – event sequence
; – process sequence
[] – selection
| | – *parallel composition*

Textual notation

- CSP (Communicating Sequential Processes)
  - Appropriate to represent concurrent and sequential actions.

```
Draw-menu    = ( select-circle? → Do-circle
                 [] select-line? → Do-line)
Do-circle    = click? → set-centre → click?
               → draw-circle → skip
Do-line      = Start-line ; Rest-line
Start-line   = click? → first-point → skip
Rest-line    = ( click? → next-point → Rest-line
                 [] double-click? → last-point → skip
```

Textual notation

- CSP (Communicating Sequential Processes)

Bold-toggle	=	select-bold? → bold-on → select-bold? → bold-off
Italic-toggle	=	select-italic? → italic-on → select-italic? → bitalic-off
Under-toggle	=	select-under? → under-on → select-under? → under-off
Dialog-box	=	Bold-toggle || Italic-toggle || Under-toggle

Dialog notations

- Summary
  - Graphical
    - STN, Petri networks, JSD, Flow charts
  - Textual
    - Grammars, production rules, CSP
  - Characteristics
    - Event-based / state-based
    - Clarity / power
    - Sequential/concurrent activities

Implementation

Semantic/presentation separation

- Application ⟷
  - Operations
  - Data
- Interface
  - Components
  - Graphics
  - I/O

- Users interact with the interface.
- Their actions should be communicated to the application.
- The application should respond accordingly.

Semantic/presentation separation

- Separation between application semantics and presentation
- Improves
    - Portability – to allow the same application to be used on different systems.
    - Reusability – components reusability → cost reduction.
    - Multiple interfaces – several different interfaces can be developed to access the same functionality → + flexibility
    - Customization – interface can be customized by both the designer and the user to increase its effectiveness without having to modify the underlying application.

Design patterns for GUI

- Model-view-controller
- View tree
- Listener

Model - View - Controller

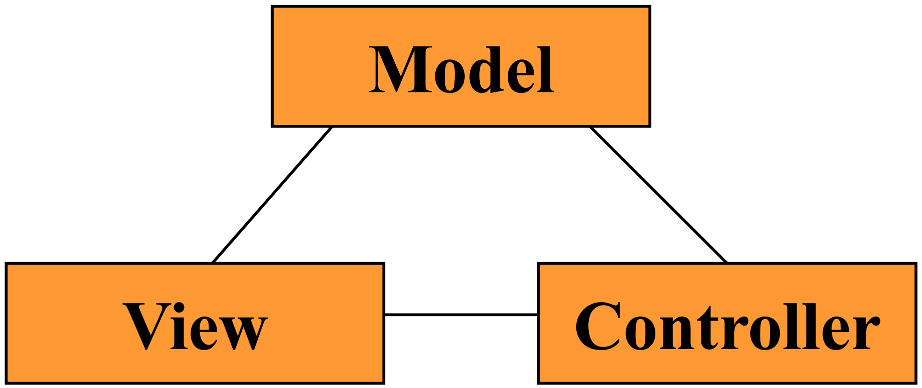
- Interactive applications architecture developed for Smalltalk
- Used by Java Swing UI widget library

  http://www.javaworld.com/javaworld/jw-04-1998/jw-04-howto.html

Model – represents the application semantics; application state and behaviour.

View – Manages the graphical and/or textual output of the application (output).

Controller – controls and manages the input (user interaction).



Model - View - Controller

- Separates frontend concerns from backend concerns.
- Separates input from output
- Permits multiple views on the same application data
- Permits views/controllers to be reused for other models

- A single model can be associated with several MVC triads, so that the same piece of application semantics can be represented by different input-output techniques.

Model - View - Controller

- Example: button
  - Model – a boolean – on or off
  - View – an image ≠ for each possible state
  - Controller – tells the model to change the state and tells the View to update the output.

- Application/presentation Separation
  - Modifications and maintenance are more easy
    - Different look → change View, nothing else

Model - View - Controller

- Model
  - Responsible for data
    - Maintains application state
    - Implements state changing behaviour
    - Notifies dependent views/controllers when changes occurs
- View
  - Responsible for output
    - Occupies screen (position, size)
    - Draws on the screen
    - Listens for changes to the model
- Controller
  - Responsible for input
    - Listens for keyboard and mouse events
    - Instructs the model or the view to change accordingly

Model - View - Controller

- View and controller are tightly coupled
  - intense communication between them
  - pairs view/controller

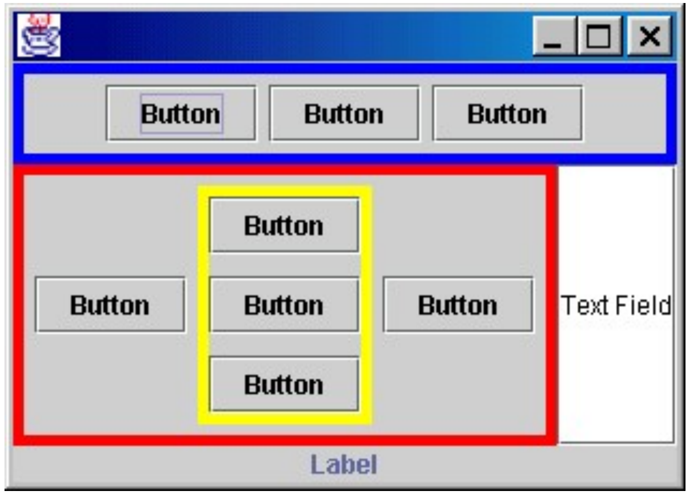
- The MVC pattern has been replaced by the MV (Model-view)

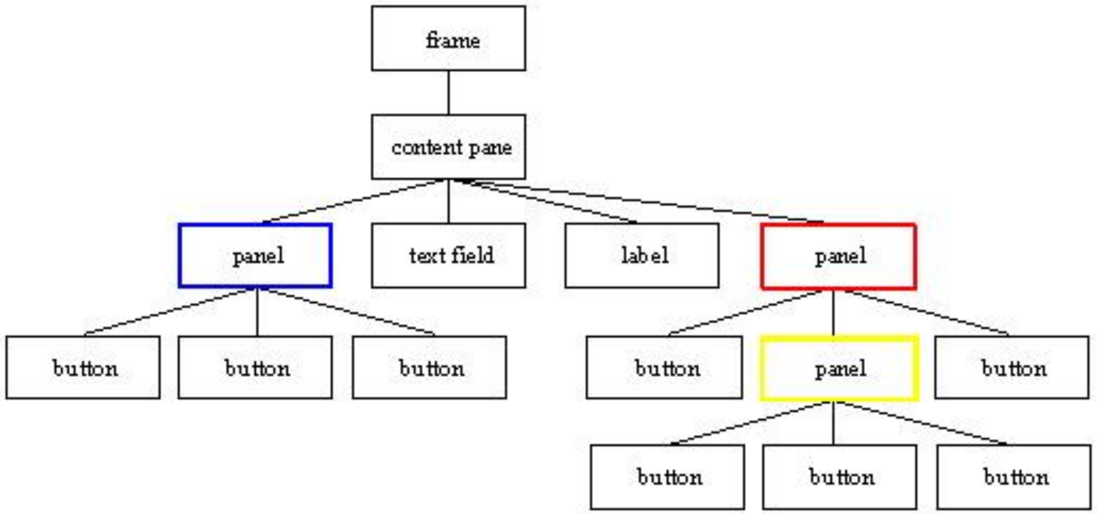
- A widget is a reusable object that manages both its input and output.
  - Also called components (Java) or controls (windows)
  - Examples: buttons, scrollbar

View tree

A GUI is structured as a tree of views

- A view is an object that displays itself on a region of the screen





View tree

- Input
  - GUIs receive keyboard and mouse input by attaching listeners to views.
- Output
  - GUIs change their output by changing the view tree
  - A redraw algorithm automatically redraws the affected views.
- Layout
  - Automatic layout algorithm traverses the tree to calculate positions and sizes of views.

Input handling

- Input handlers (listeners, event handlers,...) are associated with views
  - handle mouse input: attach a handler to the view that is called when the mouse is clicked on.

Listener pattern

- GUI input event handling is an instance of the Listener pattern.
- An event source generates a stream of discrete events (ex: Mouse events)
- Listeners register interest in events from the source, providing a function to be called when a new event occurs.
- When an event occurs, the event source distributes it to all subscribed listeners, by calling their callback functions.

Separation of concerns

- Input from the output
  - Output is represented by the view tree
  - Input is handled by listeners attached to views

- Frontend from the backend
  - Backend (model) represents the actual data that is shown and edited through the user interface.

•

References

- Dix, Alan, Finlay, Janet, Abowd, Gregory, Beale, Russel. *Human-Computer Interaction*. Prentice Hall Europe, London, 1998.