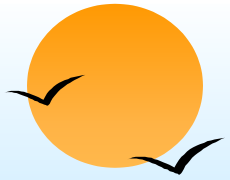
Statecharts Based GUI Design

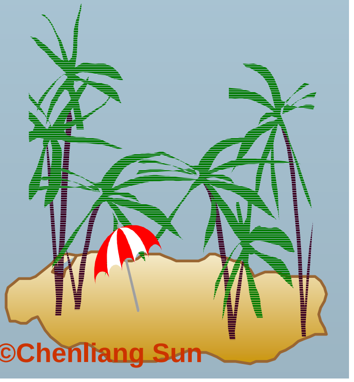
Chenliang Sun
csun1@cs.mcgill.ca
School of Computer Science
McGill University

March 5, 2003



Overview

- What's GUI ?
- Why GUI ?
- Why Statechart Based GUI Design ?
- What's Statechart ?
- How ?
- Case Study
- Testing and Coding
- Conclusion
- References



What's GUI

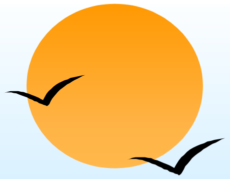
- User Interface

  * Command Line Interface

  * Graphical User Interface (GUI)

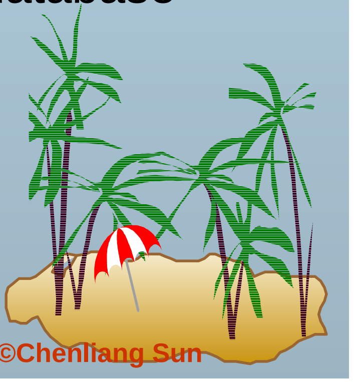
  * Hybrid User Interface

- UI: ensure a user can only supply valid events to a UI and that correct actions are executed in response to each event.



Why GUI

- Command line interfaces are difficult to use because they require users to learn a command language in order to interact with the system.
- It will be impossible for user that don’t know the command language to interact with the system. e.g. Unix shell, SQL, not good for casual or novice users
- GUI: user does not need to know command language.
  - ★ Individual UI objects, such as buttons, scrollbars and windows are combined to represent entities, such as file systems or database tables.



Why GUI

- A good GUI will allow a user with no previous knowledge of the interface to carry out useful and meaningful dialogue with system.

- In short:

  GUI allows: Direct manipulation of objects via “Pointing and Clicking” replace much of the typing of the arcane command.

Why Statecharts Based Design?

- GUIs are intrinsically far more complicated than command line interfaces because a user can have several partially completed dialog that can be suspended and resumed at any time.
- GUI must ensure that a user can only perform valid operations. e.g. Rename => name of file not blank
- GUIs contain more bugs and are usually more difficult to test and enhance than other types of code in a system
- ……
- But we have a lot of powerful and sophistic tools !?

Why Statecharts Based Design?

- Traditional GUI Design:

1. event-action paradigm

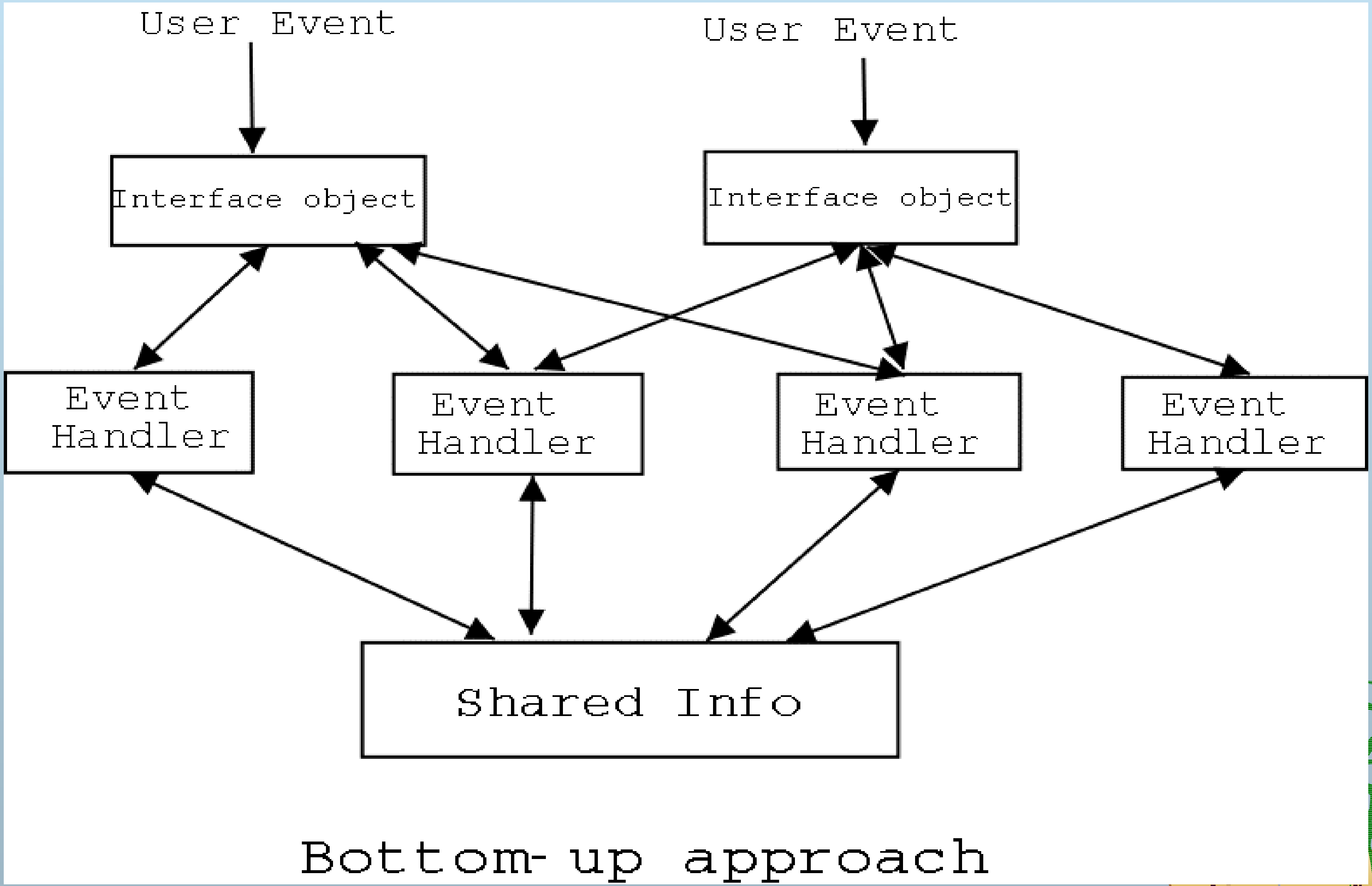
   Idea: each event supplied by a user determines which actions are executed in response to that event

   - could not accurately describe the structure of UI code. The actions that execute in response to user event (UE) affected by the context in which they occur. i.e. the UI objects that appear in an application must be coordinated to work together as a whole.

   → make use of global information

Why Statecharts Based Design?

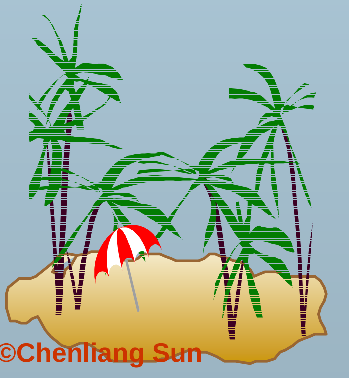
- Bottom-up approach



Why Statecharts Based Design?

- Traditional GUIs design: Bottom-up approach
  (We will explain it in more details later)

1. The code can be difficult to understand and review thoroughly
2. Difficult to test in a systematic and thorough way
3. Contain bugs even after extensive testing and fixing
4. Be difficult to enhance without introducing unwanted side-effects
5. ……



Why Statecharts Based Design?

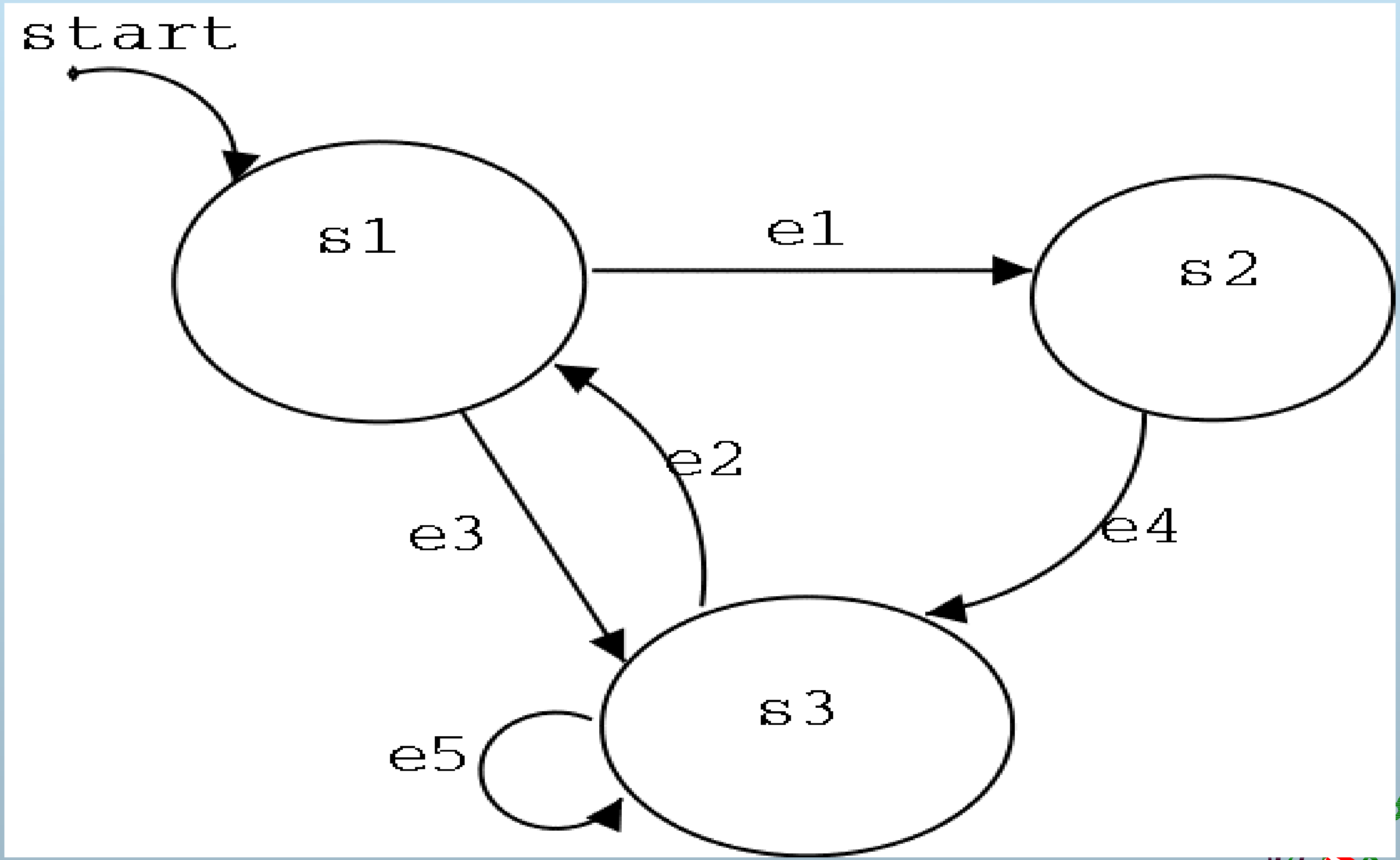
2. event-state-action paradigm

- Idea:

```
initialization
while (!quit) {
    enable selection of commands objects
    wait for user selection
    switch (selection) {
        Process selection to complete command
                or process completed command,
        Update model and screen as needed
        }
}
```

Why Statecharts Based Design?

- FSA



$$\begin{tikzcd}
\text{start} \arrow[d] & & \\
s1 \arrow[rr, "e1"] \arrow[dr, "e3"'] & & s2 \arrow[dl, bend left, "e4"] \\
& s3 \arrow[ul, bend right, "e2"'] \arrow[loop left, "e5"] &
\end{tikzcd}$$

Why Statecharts Based Design?

- The event that a user supplies will cause the software to move from one state to another.
- The states define the context in which an event occur
- It's the natural of all direct manipulation UI
- But

  - large number of states and events arrows for large, complex system

  - The state transition diagram was large and difficult to read, to draw, to change

  - not scalable

  - non-user friendly

Why Statecharts Based Design?

- Using Statecharts:
  - ★ much more powerful and expressive state based language
  - ★ Raise UI development from a coding task to a software design task

+ can be written quickly and easily. Even more:

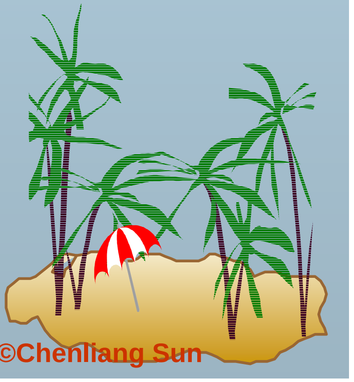
AUTOMATICALLY CODE GENERATION !!!

+ easy to test using white box techniques

+ easy to enhance repeatedly over the lifetime of a system

+ can be modified without introducing unwanted side-effects

+ can be regression tested without the need for full re-test



Why Statecharts Based Design?

Really?

Before answer this question, we first should answer another question: What’s Statechart ?

What's Statechart?

- Proposed by David Harel (1987)

  Statecharts: A visual formalism for complex systems
- *Provide a very rich and expressive notations that allows complex system to be specified concisely and at different levels of abstraction*
  - ★ Hierarchy –Systems designed as a hierarchy are generally easier to understand because the structure of the system is not obscured by irrelevant details
- The history mechanism is used to remember the last state that the statechart was in within a particular set of states. Thus on returning to the set of states, the most recently visited of states in the set will be entered.
- A statechart is faster to create and easier to understand than the words for the same information.

What’s Statechart?

- It is easy to achieve a complete specification of a UI using statechart because it is easy to spot all the scenarios that need to be specified
- A user interface can be decomposed into modules and each module can be specified independently of each other

How to design GUI based on Statecharts?

- There is no standard recipe to develop software, including GUI
- Depend on the problem / requirement AND experience of programmer
- Following method is a suggestion:

  Design heuristics

How

- 1. Identify the high-level statechart
  - ★ The first step in most designs is to identify the high-level states in the design. These states usually corresponding to screens (Canvas) that a user can navigate between.
  - ★ It is good to know which user events will cause the UI to move between different canvases early in the design process.
  - ★ This allows the UI to be divided into parts that can be constructed by different developer.
  - ★ Problem related to integrating screen can be avoided

Hint: Design one screen at a time, but start to understand how two screens interact reasonably early in the process

How ?

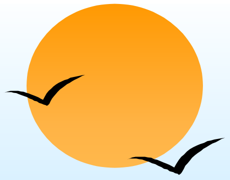
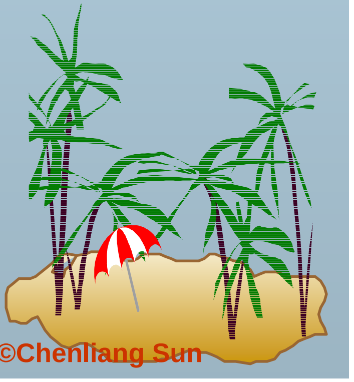
- 2 Identify screen rules
  - ★ Write down all the items in or associated with a screen, buttons, menu items, text items, radio buttons, scrolling lists, keyboard shortcuts, etc
  - ★ Identify whether the behavior or appearance of an item is constant or whether it varies
  - ★ Identify behaviors of items

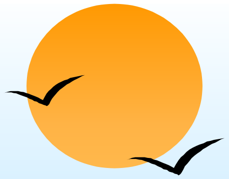
    e.g. when they are available to a user

    scroll list delete, sub window for detail

    How about no element in the list?

Let’s talk it in more detail



How ?

- 2 Identify screen rules

(1) Entry and exit rules

what user events cause a screen to be entered or exited? Start application? Close application?

(2) Identify any modes

A mode is a state that the software can enter and where the effect of a user event changes depending on the mode

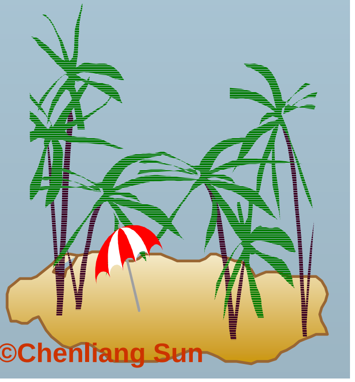
e.g.. Drawing package: pencil, spray or paint brush?

read-only? Read / write?

(3) Identify the screen items that have varying behavior

enable/disable, visible/invisible, color

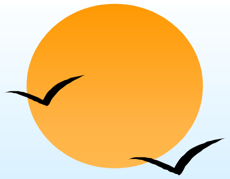
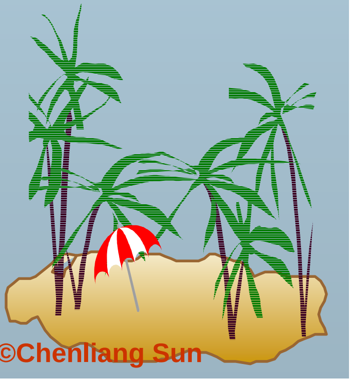
(4) Identify the screen items that have constant behavior



How

- 3 Identify states: converting screen rules to a statechart.
  - ★ Draw a statechart that defines the required behavior of each screen item as specified in the screen rules and ensure the screen items work together as a whole
  - ★ Think it in an abstract way, i.e. identify the states first. This builds a solid framework on which events and actions can easily be added.
  - ★ Note:

    Any type of user interface can be specified in terms of events, conditions, states and actions

How ?

- 4. Consolidate related behavior

  Look for common events and conditions that will cause a number of screen items to change state. What do these states represent? Can related screen items be brought together into the same states?

  ex. Cut, copy, paste can be performed by menu items, icon buttons, and key press, then these related functions can be controlled by the same part of the statechart

- Separate unrelated behavior

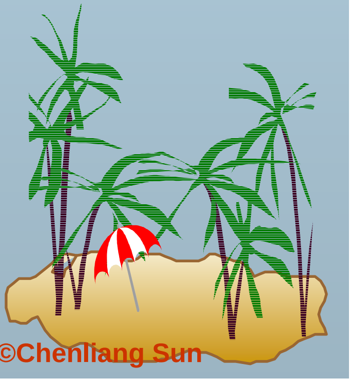
  use depth extensively to structure a statechart

How ?

- 5. Keep dependencies out of independent concurrent parts
  - ★ The use of dependencies weakens a design and should therefore only be used to as a last resort
  - ★ Dependencies make testing more difficult
  - ★ Dependencies also make the long-term maintainability of software more difficult because changing one concurrent part may cause an unwanted side-effect in another part

Unavoidable dependencies?

Consider merging them



How

- 6 Synchronize concurrent parts with simultaneous events
  - ★ It is possible for a single user event to cause simultaneous state transitions in concurrent parts

    e.g. delete an item from a list by clicking a delete button, this event may cause a transition which results in the action of deleting the item from the list. The same delete event may also cause a simultaneous transition in a concurrent part to a state which an undo button is enabled

Hint: when designing simultaneous events, typically only one event should cause actions to occur. The other events should be used to synchronize the concurrent parts with the primary event arrow..

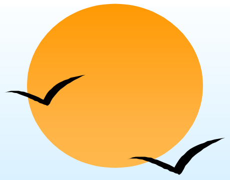
How

- 7 Be wary of actions on states
  - ★ Actions on states are like global variables. They cab be very powerful, but they also have potential to cause problems.
  - ★ Hint:

    use states to control the attributes of UI items rather than executing actions. All actions should be associated with events.

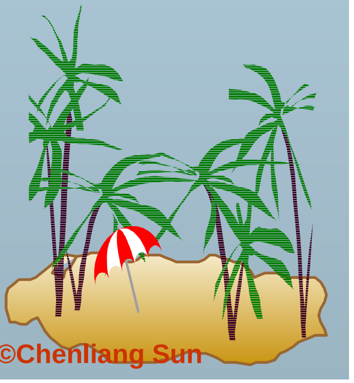
How ?

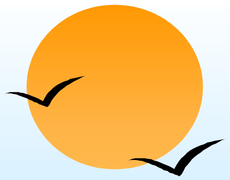
- 8. Avoid event arrows that transcend many levels in a state hierarchy
  - ★ To understand the statechart without having to understand the details of lower-level states



How ?

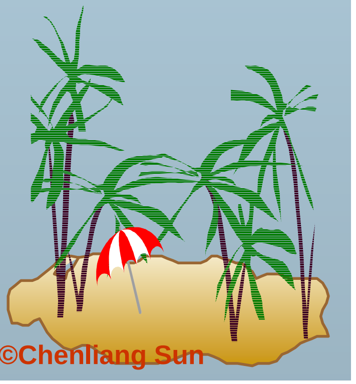
- 9. Naming states
  - ★ Each state should be given a meaningful name. If you cannot think of a name that conveys what the state represents then this may suggest there is a problem with the design.
  - ★ If the meaning of the state cannot be identified in the design process then how will it be understood during the maintenance of the system?
  - ★ Each state must explicitly represent something





How

- 10. the route to a state should be irrelevant
  - It should never be assumed that a previous state has defined an attribute of a UI object and also does not need to be defined again in current state



How ?

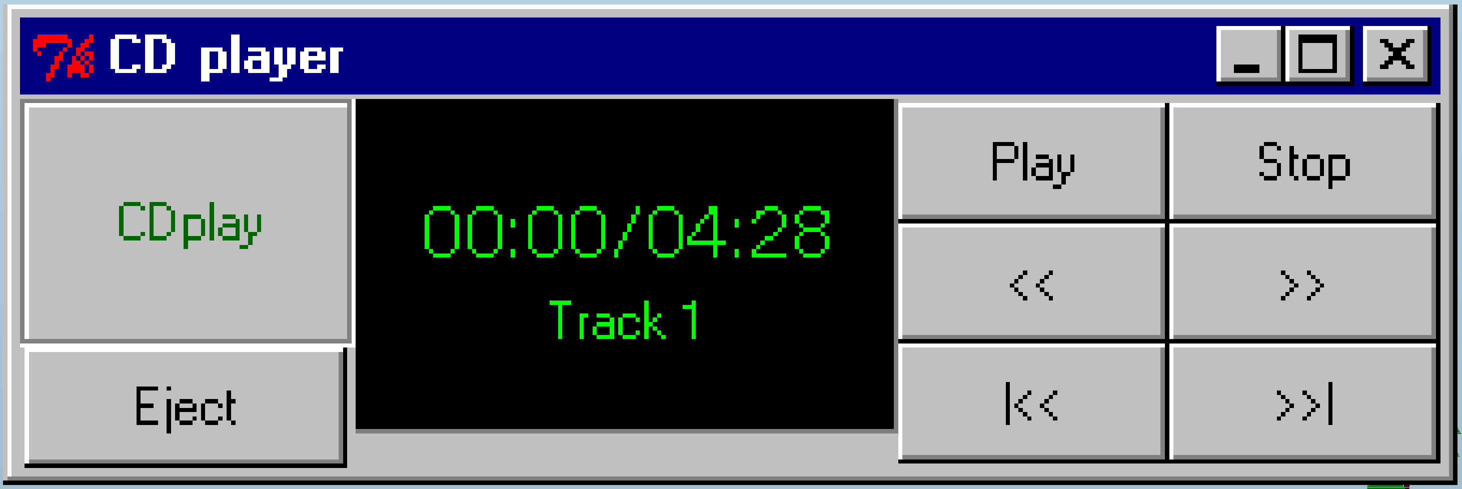
- 11. Avoid convoluted conditions
  - ★ The conditions associated with an event can sometimes be simplified by using priorities.

Case Study : CD Player

High-level Requirements:

http://moncs.cs.mcgill.ca/people/hv/teaching/MS/assignments/assignment3/

Overview of the user interface screen



Case Study : CD Player

- Screen Rules
  - **Entry and exit events**:

    The screen can be entered by starting the application

    The screen is exited when Close button is clicked

    There is no explicit Quit button by requirement
  - **Modes**

    When in the CD Stopped state, the Stop button shall be disabled.

    When in CD playing state, the button shows “Pause”

    But when in pause state, the button shows “Play” (not required)

Case Study : CD Player

- Screen Rules
  - **Items with varying behavior**:

    When in the CD Paused state, the values in the Time and Track fields will be displayed initially and then after one second will be hidden (blank). After another second, they will be displayed again. This displaying/hiding cycle will continue as long as the system is in the CD Paused state

    Display text will change according to states
  - **Items with constant behavior**:

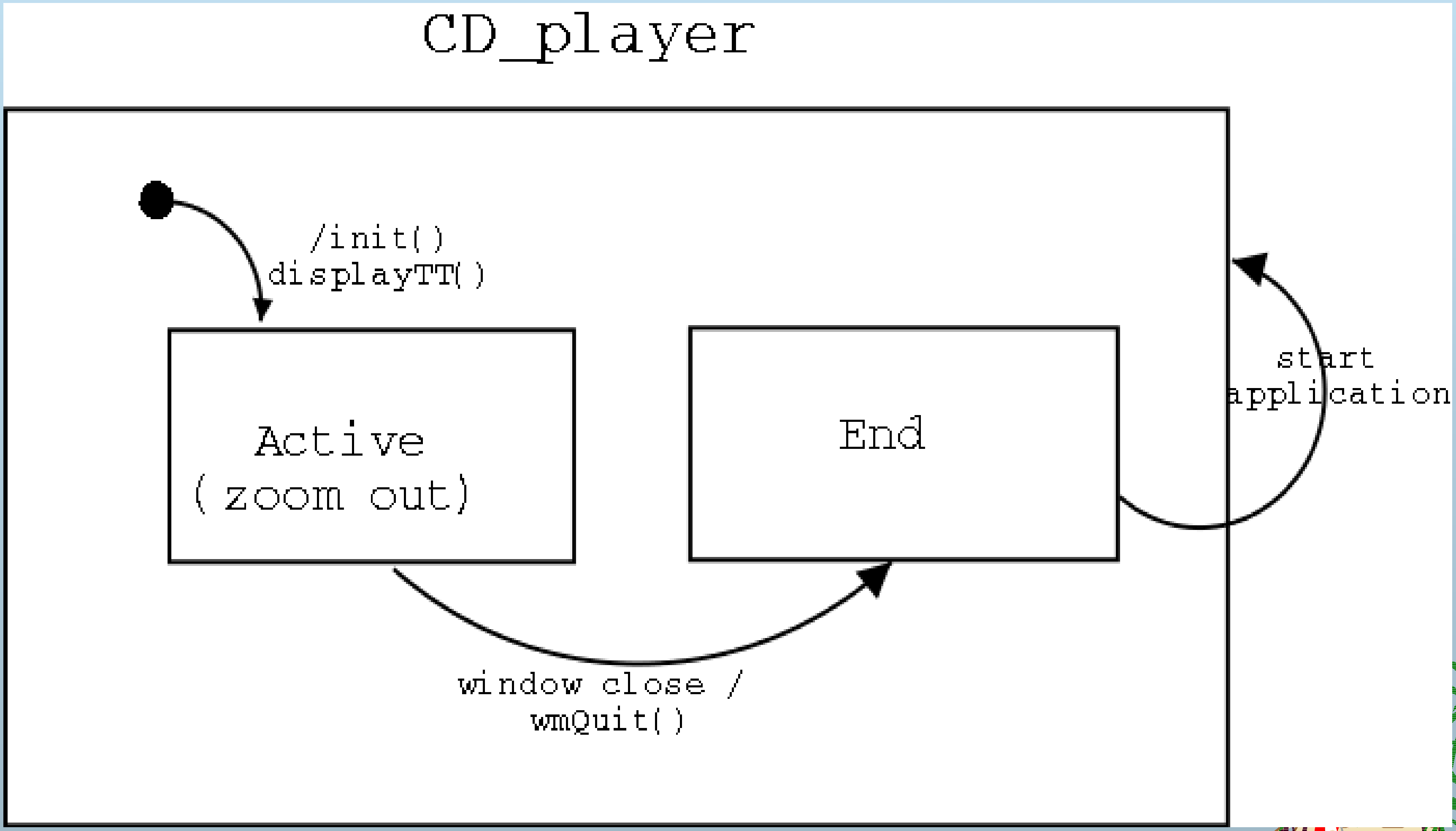
    the Close button is always enabled

    the Eject button is always enabled. i.e. At any time the user can eject the CD player

    When in the CD Stopped state, the Time field shall display 00:00 and the Track field shall display [track 1].

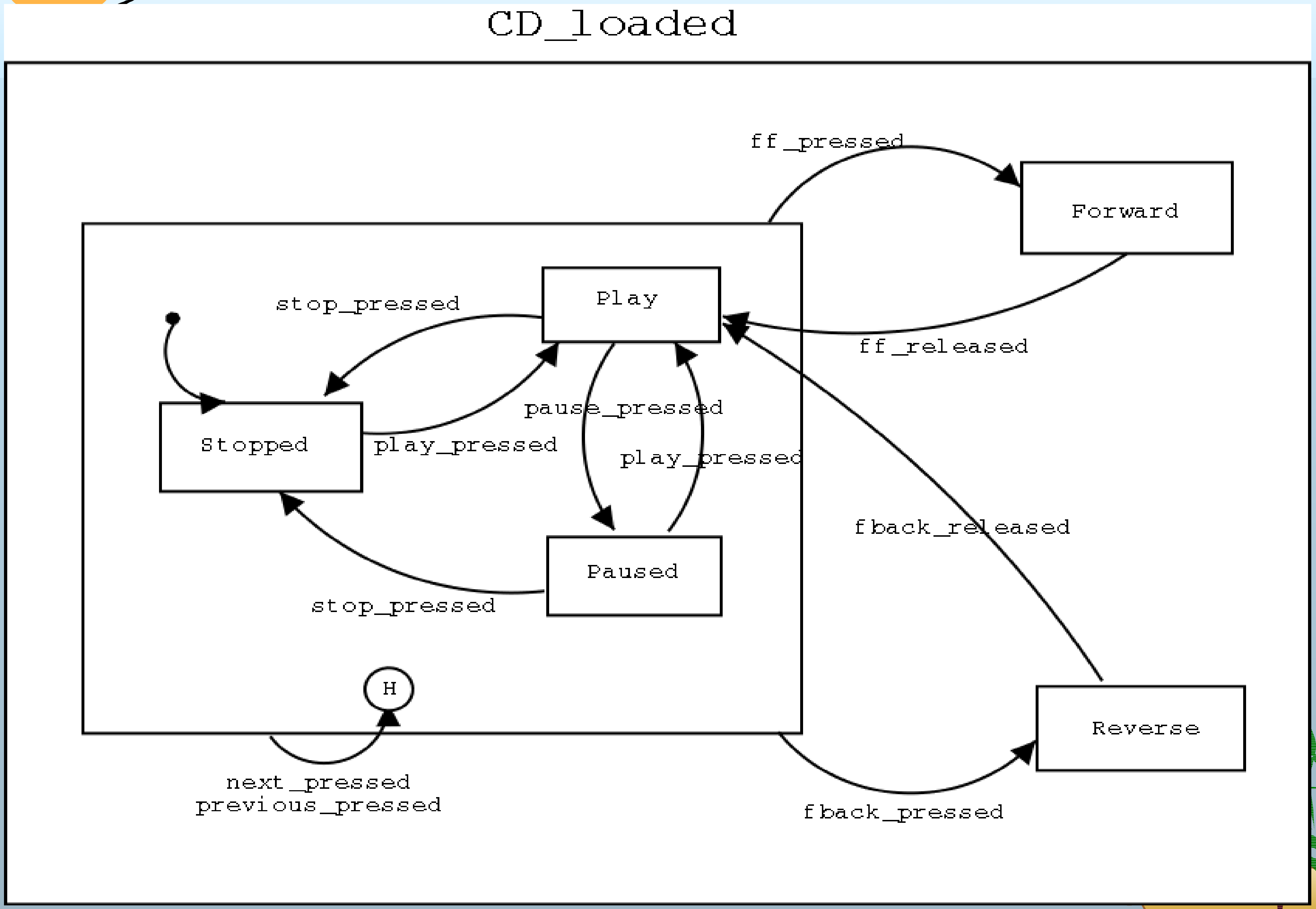
Case Study : CD Player

- The high-level statechart



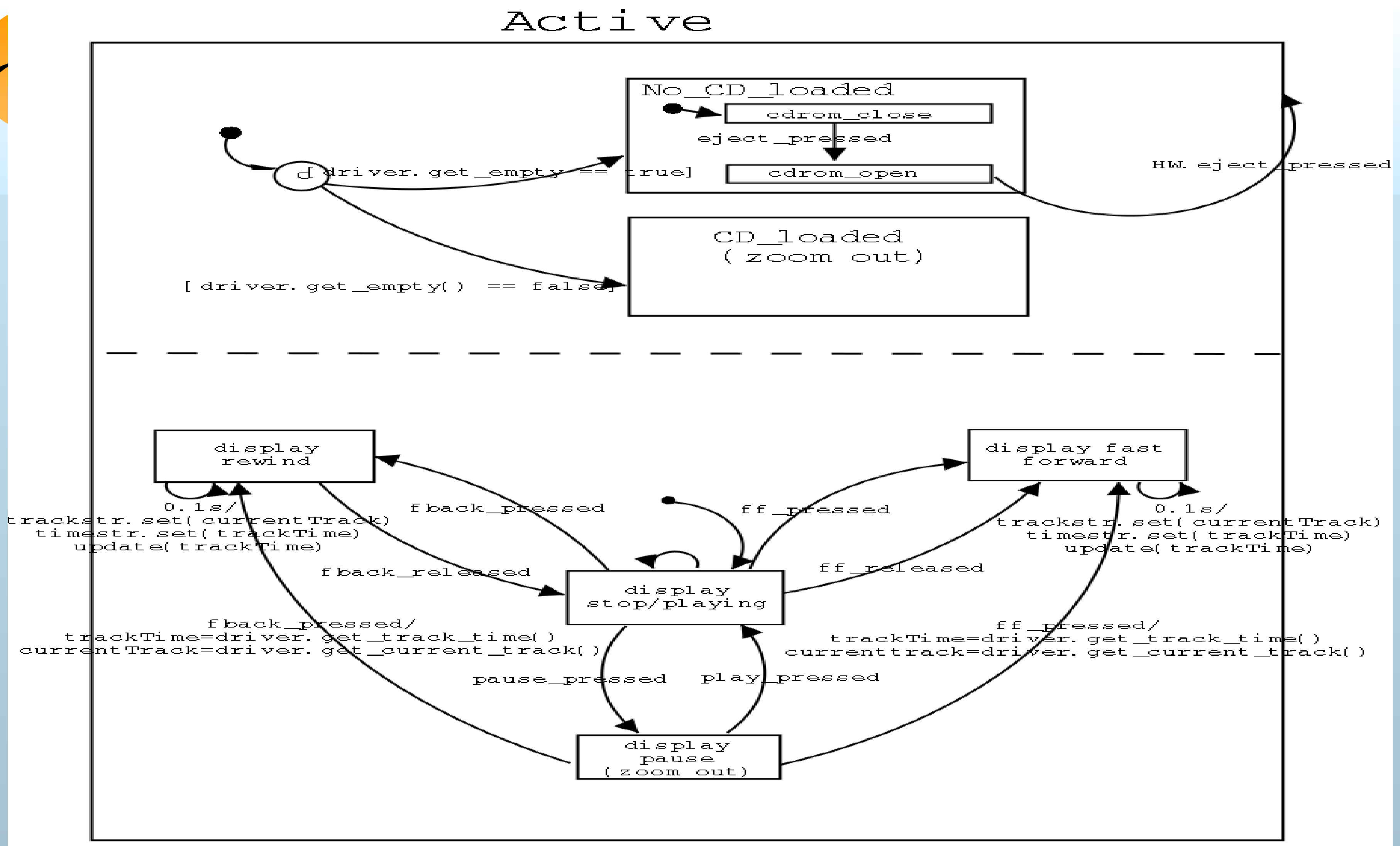
Case Study : CD Player

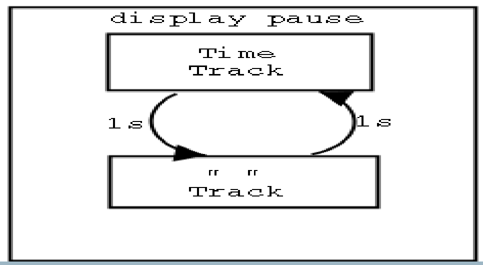
- Convert screen rules to a statechart


CD_loaded
ff_pressed
Forward
stop_pressed
Play
ff_released
Stopped
play_pressed
pause_pressed
play_pressed
fback_released
Paused
stop_pressed
H
Reverse
next_pressed
previous_pressed
fback_pressed

Case Study : CD Player

- Synchronize concurrent parts with simultaneous events statechart


Active
No_CD_loaded
cdrom_close
eject_pressed
cdrom_open
d
[driver.get_empty == true]
HW.eject_pressed
CD_loaded
(zoom out)
[driver.get_empty() == false]
display
rewind
display fast
forward
0.1s/
trackstr.set(currentTrack)
timestr.set(trackTime)
update(trackTime)
fback_pressed
ff_pressed
fback_released
ff_released
display
stop/playing
fback_pressed/
trackTime=driver.get_track_time()
currentTrack=driver.get_current_track()
ff_pressed/
trackTime=driver.get_track_time()
currenttrack=driver.get_current_track()
pause_pressed
play_pressed
display
pause
(zoom out)


display pause
Time
Track
1s
1s
" "
Track

Case Study : CD Player

- Keep dependencies out of independent concurrent parts
- Be wary of actions on states
- Avoid events that transcend many levels in a state hierarchy
- Naming states
- ……

Coding a statechart

Coding a statechart is a simple process. There are 4 main tasks which should be carried out in the following order:

- Create the user interface objects (ideally this should be done by an interaction designer)
- Create state variables
- Create the state procedures
- Implement the state transitions defined in the event-action tables of the statechart

Testing Statecharts

- Test to find errors in the design and the implementation
- The state transitions should be made visible during testing
- Check the action carefully
- Check the state carefully
- Check for dead states
- Ensure events that are not supposed to be *possible*, really cannot happen

Conclusion

- Statecharts provide much more powerful and expressive state based language, which is much more concise and accurate than natural language
- Statechart based design raise UI development from a coding task to a software design task
- Code from statecharts can be written quickly and easily. Even allows

  AUTOMATICALLY CODE GENERATION !
- easy to test using white box techniques
- …….
- Interaction Object Graphs (IOGs) are an extension of statecharts, and are designed to specify the details of UI widgets. For example, IOGs allow to define new Interface objects without laboriously coding. But that is beyond this presentation.

References

- Ian Horrocks, Constructing the User Interface with Statecharts, Addison-Wesley, 1998
- Hans Vangheluwe, Modeling and Simulation course lecture notes, School of Computer Science, McGill University, 2002
- Carr, D., **Interaction Object Graphs: An Executable Graphical Notation for Specifying User Interfaces**, *Formal Methods for Computer-Human Interaction*, P. Palanque and F. Paterno', editors, ISBN 3-540-76158-6, Springer-Verlag, 141-156, Nov. 1997.