Human-Computer Dialogues

Overview

- Dialogues
  – Human-Human
  – Human-Computer
- Designing and Documenting dialogues
  – Scenarios
  – Finite State Machines
  – Production Systems

Human-Human Dialogues

- Based on threads of conversation
  - Single-threaded
  - Turn-taking
  - Conventions

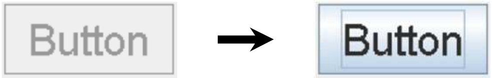
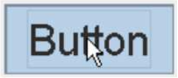
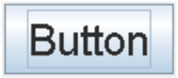
Human-Computer Dialogues

- Conventions
  - Needed for dialogues to work smoothly
  - Conventions need to be low-level/subconscious
    - Don’t want to think about them
    - Easy to learn
    - Easy to apply to new situations
    - So low-level it’s hard to imagine any other way to do it
- Draw inspiration from Human-Human dialogues, but not identical because...
  –

Human-Computer Conventions

- Prompt
  - “I’m ready to converse, if you want to.”
  - Could be many prompts available.
- Echo
  - “I’m receiving your input.”
- Accept Trigger
  - “I understand that you’re done.”
- Acknowledge
  - “I’m working on your request.”
  - Often omitted if response is immediate.
- Respond
  - “Here’s what I’ve done.”
  - Normally followed by a new prompt.

Example: Button

Prompt	
Echo	
Accept Trigger	mouse up
Acknowledge	 + ???
Response	application specific

Example: Form

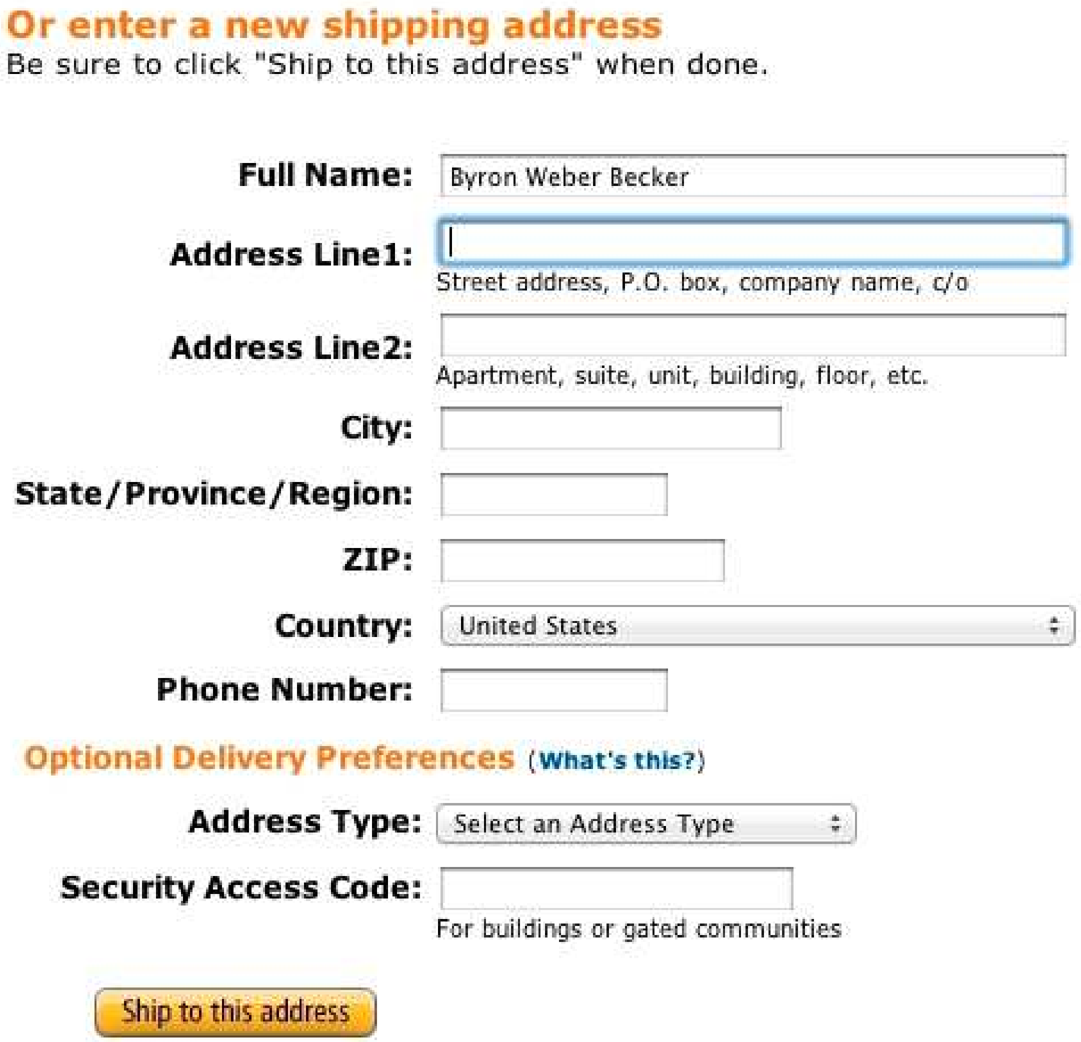
Prompt

Echo

Accept Trigger

Acknowledge

Response



Or enter a new shipping address

Be sure to click "Ship to this address" when done.

Full Name:	Byron Weber Becker
Address Line1:	Street address, P.O. box, company name, c/o
Address Line2:	Apartment, suite, unit, building, floor, etc.
City:	
State/Province/Region:	
ZIP:	
Country:	United States
Phone Number:	

Optional Delivery Preferences (What's this?)

Address Type:	Select an Address Type
Security Access Code:	For buildings or gated communities

Example: Keystroke

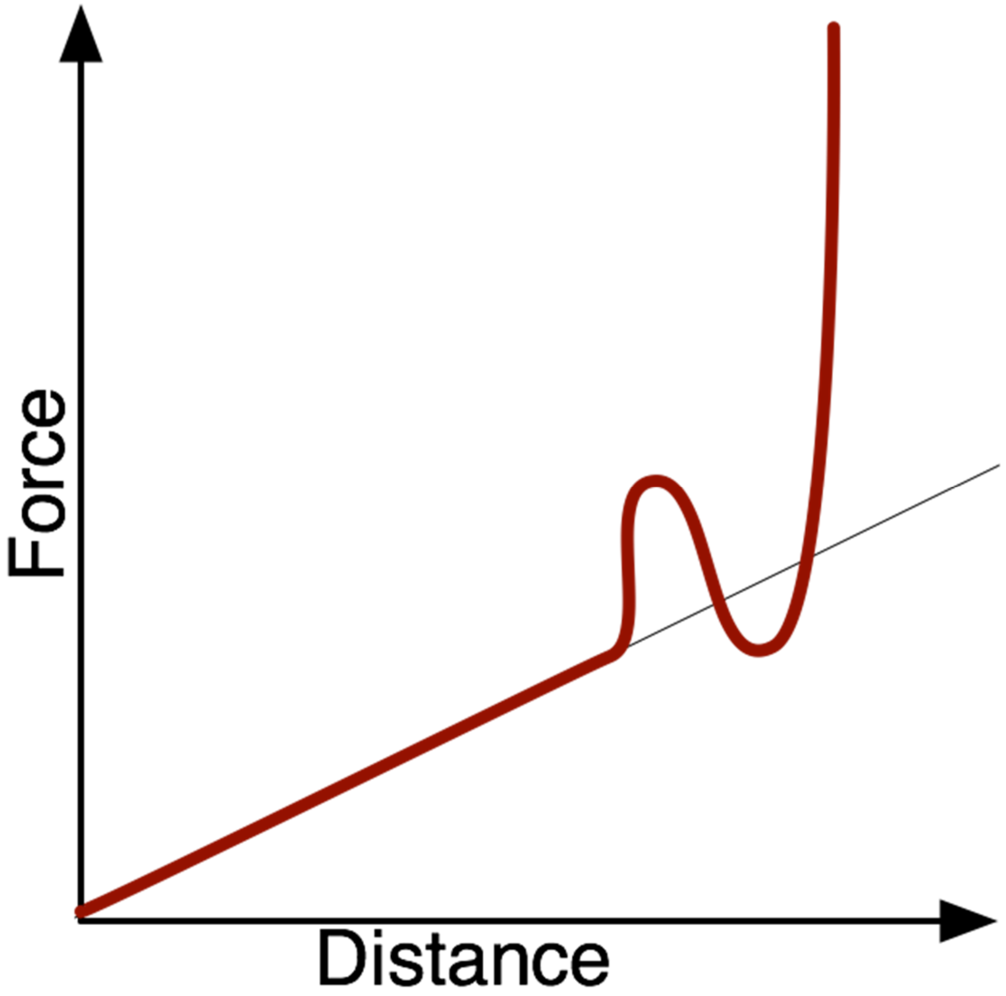
Prompt

Echo

Accept Trigger

Acknowledge

Response



Designing & Documenting Dialogues

- Important to get these dialogues “right”. Why?

- Computers are an integral part of society
  - Spectrum: entertainment to life-critical systems
  - Examples: air traffic control, power plants, nuclear reactors, medical equipment, health records, factory automation, banking, stock markets, business, …
- User interfaces provide the means to control these systems
- What does it mean for an interface to be “wrong”?

When an interface is “wrong”

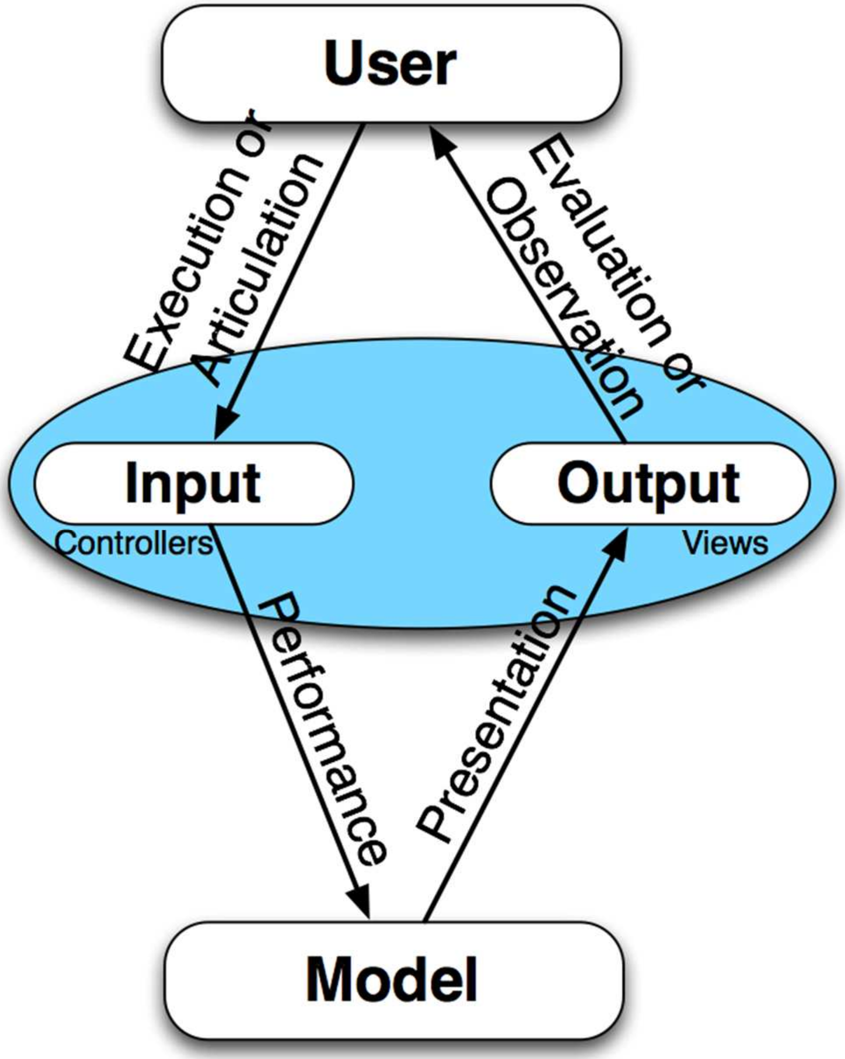
- Core dumps
- “Incorrect” calculations
- Doesn’t provide necessary features for a given task
- Doesn’t honor human’s cognitive capabilities; limitations
- Doesn’t make the operator efficient enough
- Error prone
- Doesn’t satisfy user needs
- …

Costs of being “wrong”

- Power management (Three Mile Island)
- Managing mutual funds
- E-commerce (Amazon)
- Voting systems
- Aviation
- Other examples?

Central Tension

- User: rich and varied experiences; makes intuitive leaps; learns; uses metaphors; creative
- User Interface: needs to mediate between these two radically different systems
- Model: follows a rigid program; not creative; only primitive learning (at best)



Languages

- Users use informal, natural, “open” languages
  - wonderfully extensible; can be used to write poetry and describe the smell of fresh bread
  - Ambiguous

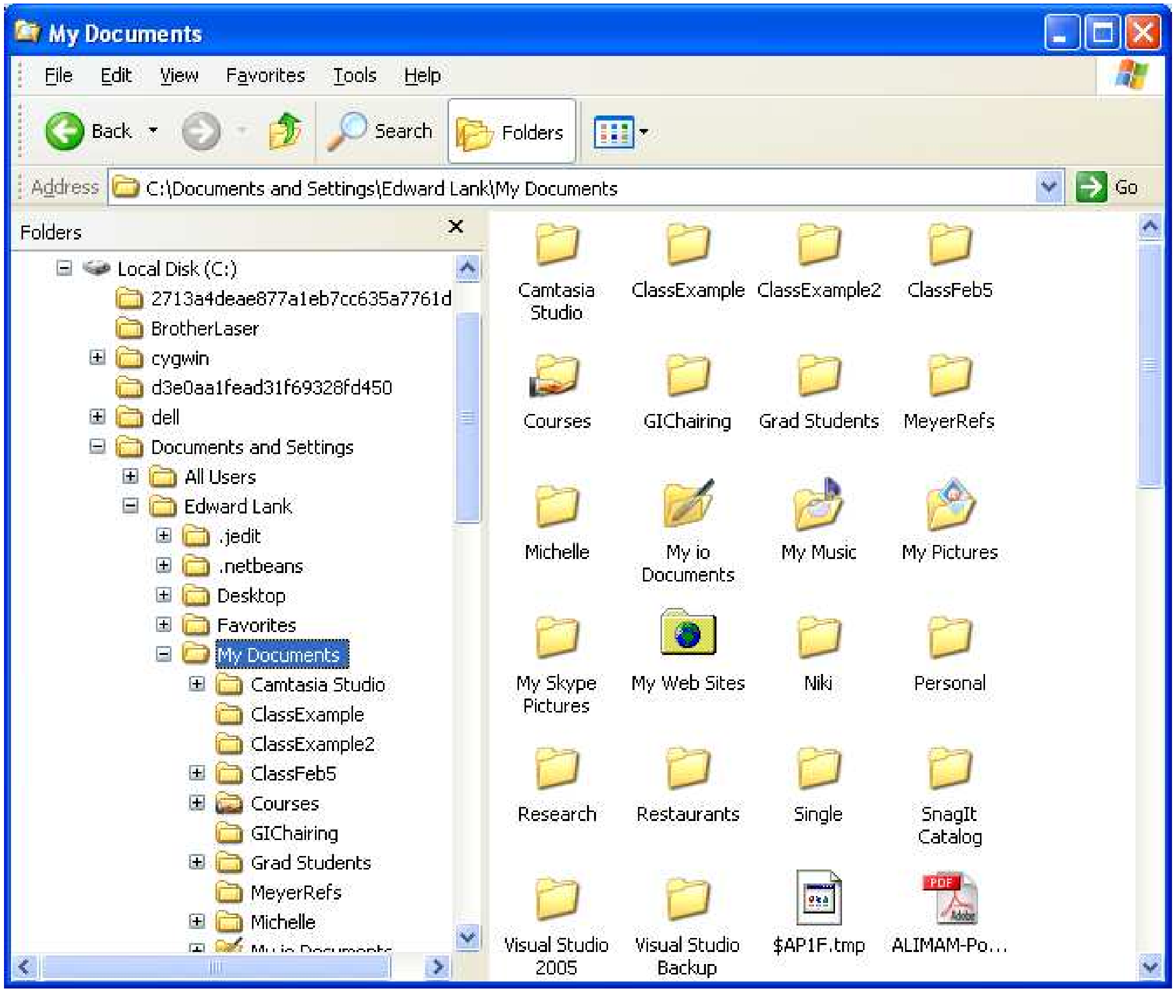
Need a language for user interfaces that mediates these two. Needs characteristics of both.

- Computers use formal or “closed” languages
  - precise terms (unlike English)
  - precise rules for combining terms
  - precise meanings for statements
  - But… Only applies to a specific domain. Can’t use it in new ways

Describing user input sequence

- Consider a button
- Click = activate
- What if
  - User presses leftmouse down outside button and drags over button, then releases on button
  - User presses leftbutton down on button, then drags off
  - User presses leftbutton down on button, drags off then back on, then releases
- Tooltips add additional complexity

More complex interfaces



Address C:\Documents and Settings\Edward Lank\My Documents

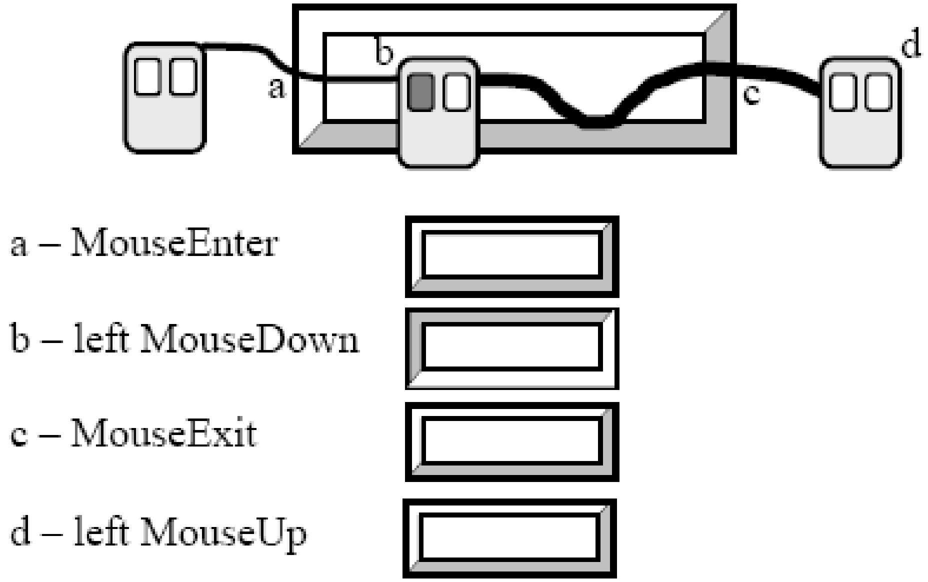
Criteria for UI "Languages"

- Understandable by users
- Easily converted into implementations
  - A closed language?
- Precise enough to settle arguments
- Facilitate answering interesting questions
  - Observability
  - Controllability
  - Pathologies

UI Language Options

- Natural language descriptions
  - “When the user clicks the mouse inside the button, fire the action event code.”
  - But…
- Mouse Event Diagrams
- Finite State Machines
- Propositional Production Systems
- Code
  - Problems?

Mouse event diagrams



a – MouseEnter

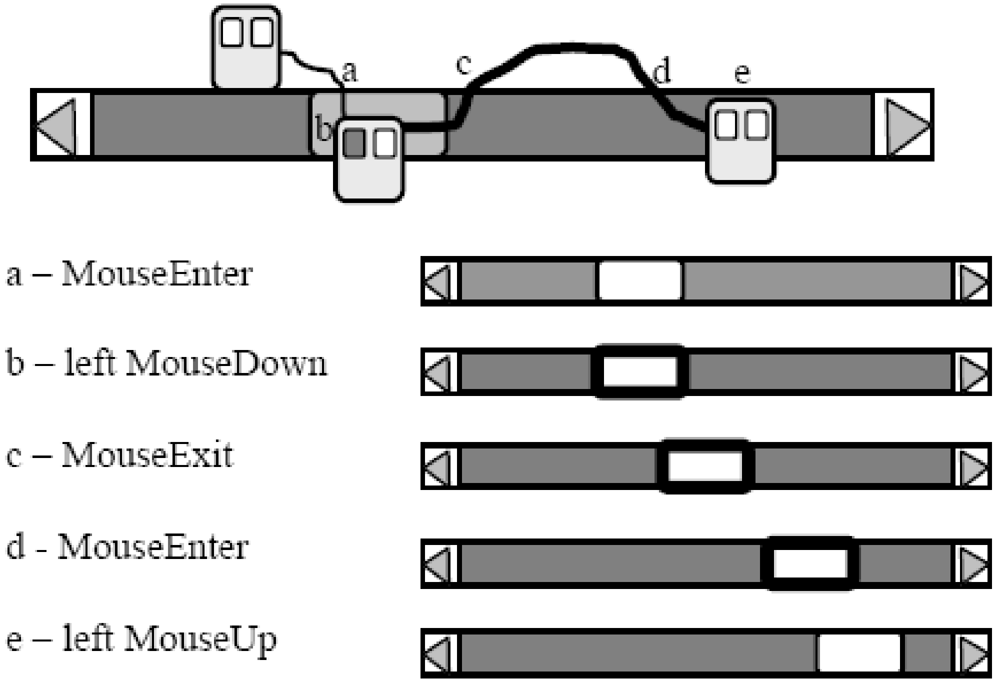
b – left MouseDown

c – MouseExit

d – left MouseUp

- Not traditional behaviour
- However
  - May occur
  - Needs to be handled by interface
- Also
  - Issue of appearance
  - On mouse-press, button appears pressed
  - If mouse moves off what happens?

Scrollbars



- More complexity
  - Down on slider
  - Dragging
  - During drag, slide off channel
  - Slide back on channel (or not)
  - Release
- At any distance?

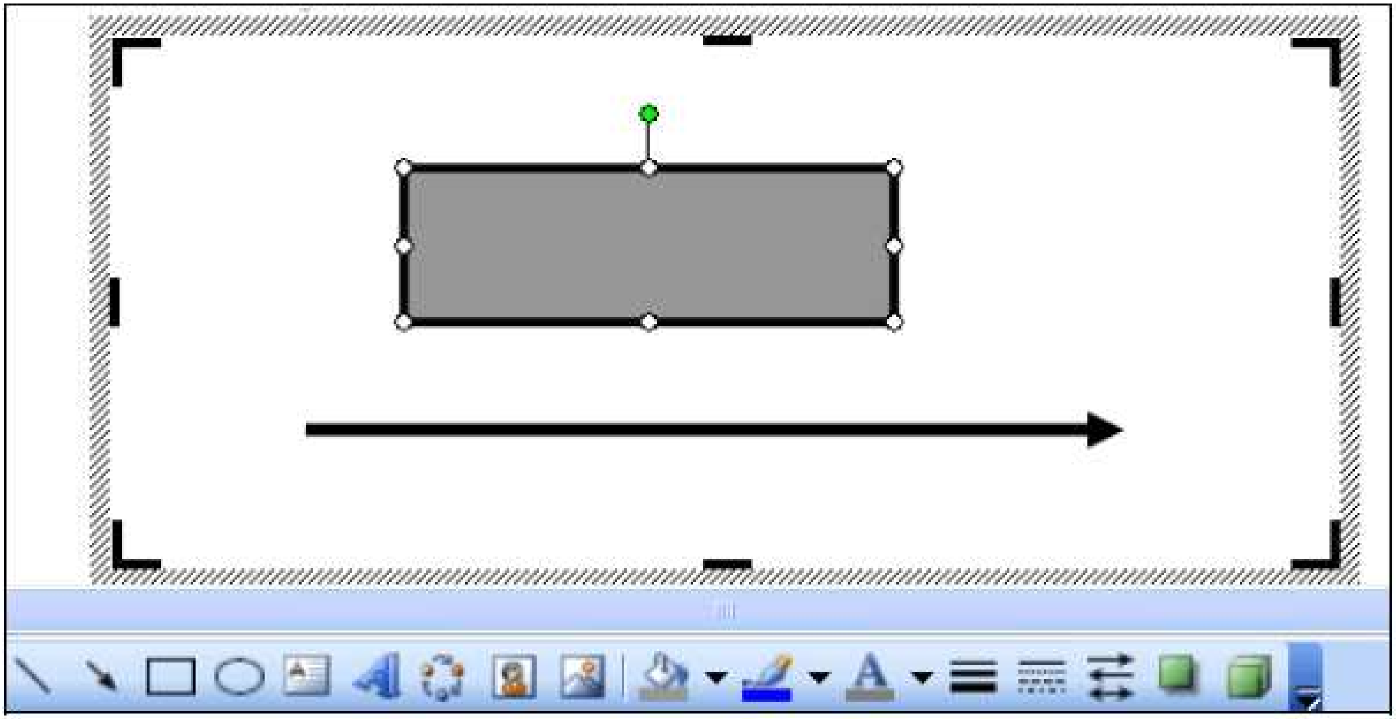
Critiquing Mouse Event Diagrams

- What are the advantages and disadvantages of MEDs?

Drawbacks of Mouse Event Diagrams

- Any widget can support many events
- Scrollbar
  - Clicking on arrows, off slider, etc.
  - Behaviour for arrows
    - Similar to button?
  - Behaviour for channel off slider
    - Similar to button?

Mouse Event Diagrams -- Drawbacks



Characteristics of Formal Languages

- Finite alphabet
- Set of well-defined formation rules
- Examples: Finite State Machines, Context Free Grammars, Programming Languages

- Characteristics
  - Precise
  - Provides an unambiguous description of part of the system
  - Limited in their expressivity; "Closed"

Benefits of Formal Languages

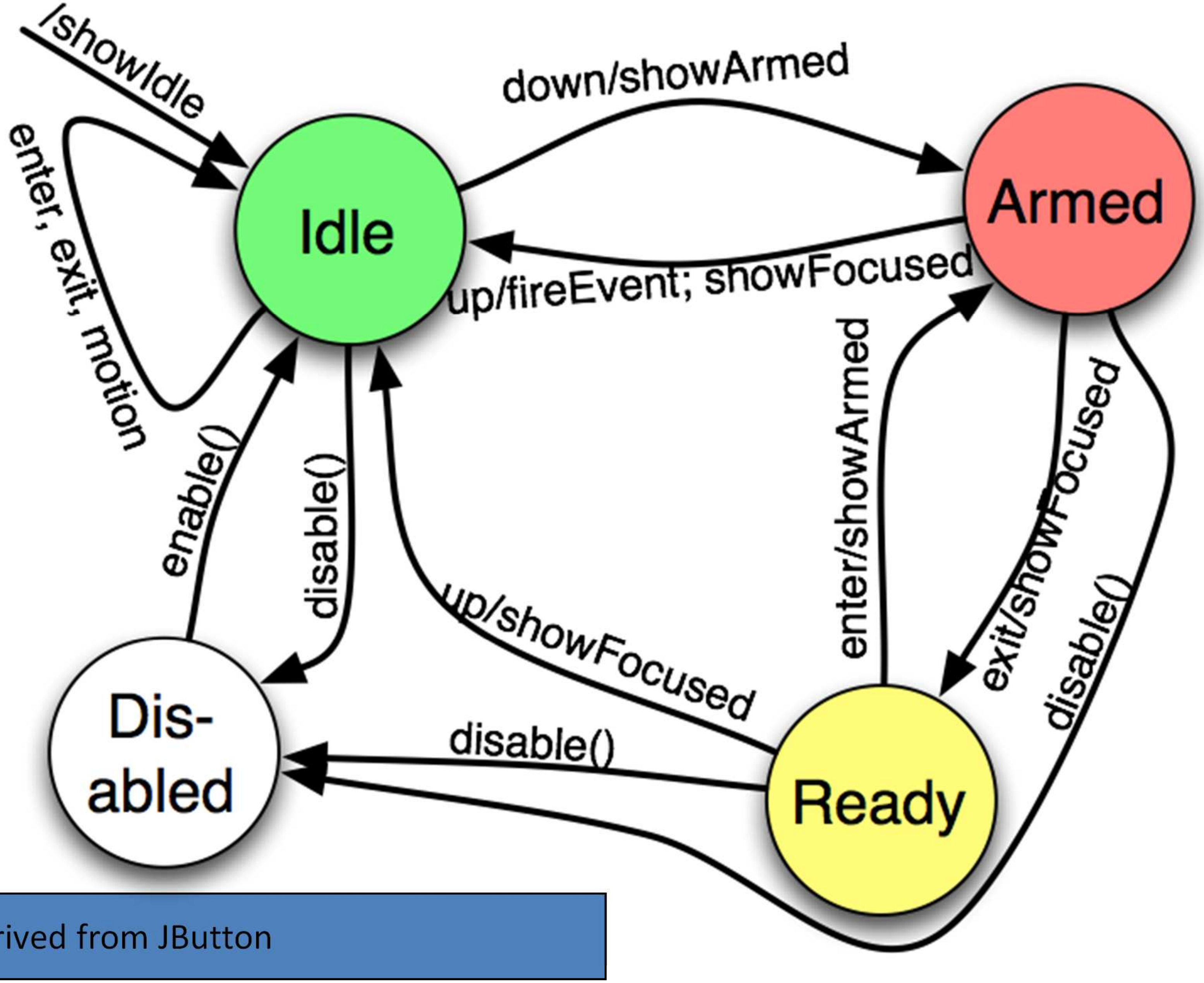
- Formalized descriptions clarify intentions:
  - Provide unambiguous description of a part of the world
  - Focus the design process
  - Define terms, concepts
  - Uncover holes in the understandings
  - Eliminate “hand waving”
  - Concrete product that others can critique and improve
  - Can suggest ways to test the system
- Descriptions provide a source of training for users:
  - Teach users how to use an interface
  - Common to write the manual before the software

Formal Languages in UIs

- What is it about an interface design that must be defined?
  - Key challenge is to define the interaction
  - How does the state of the system change as a function of user input?
- Other systems:
  - Input, process, output
  - Each part is fairly heavy-weight
- UI:
  - Input, react, input, react, input, react, …

Finite State Machines

- One possible UI formalism
- Consider a button:
  - What are the relevant states?
  - What are the transitions?
  - What are the actions taken on each transition?



Derived from JButton

Definitions: Events

Event	Meaning
down	User presses the mouse button while component has focus.
up	User releases the mouse button while the component has focus.
enter	Mouse enters the component.
exit	Mouse exits the component.
enable/disable	When in the disabled state, the user can’t interact with the button at all. It’s enabled and disabled programmatically.

Definitions: Actions

Action	Meaning
showIdle	Button interior turns light gray.
showArmed	Button interior turns dark gray.
showFocused	Button border turns blue; interior turns light gray.
fireEvent	Fire the listener(s) associated with this button.

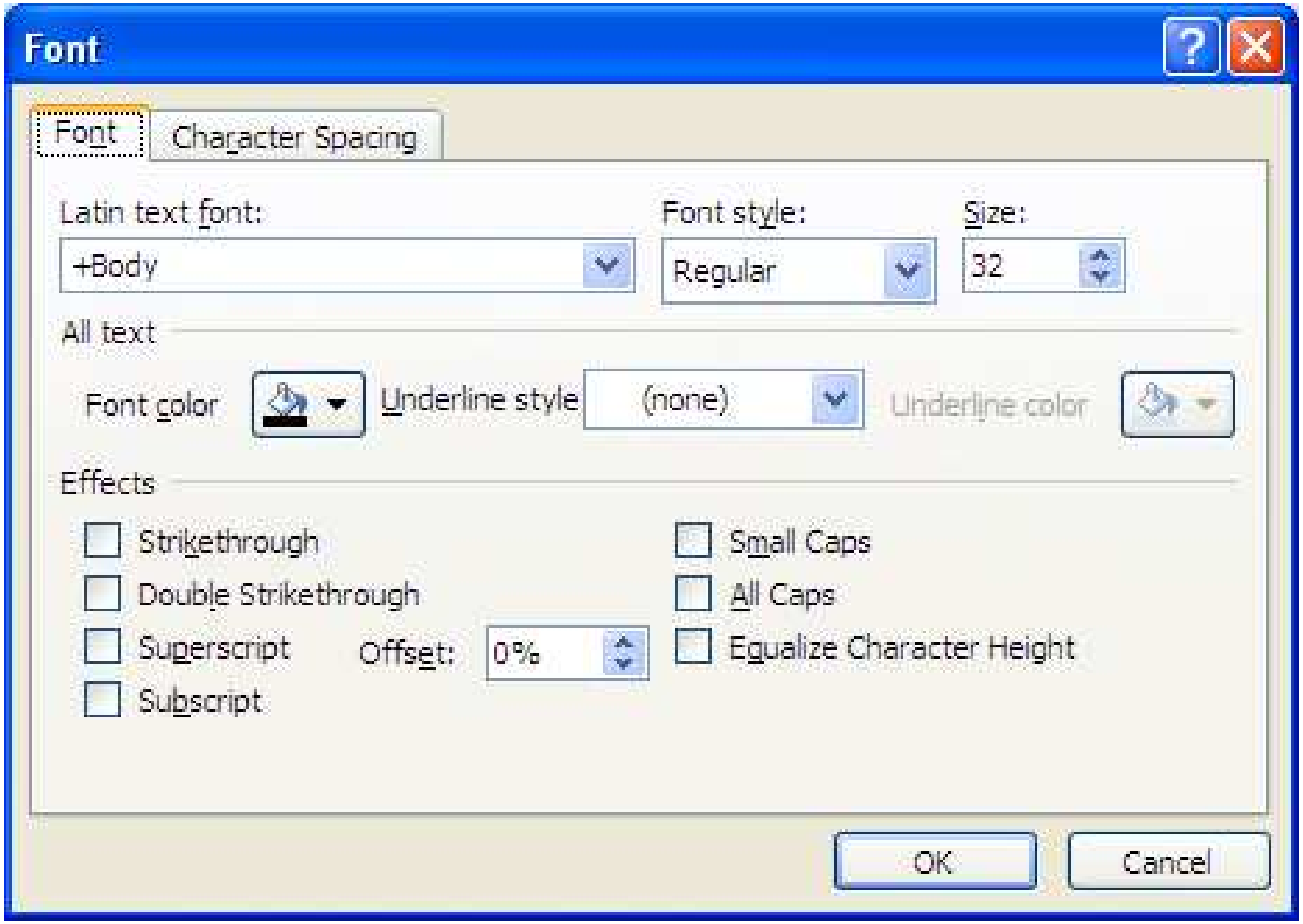
FSMs and UIs

- A well-known and understood method for formally describing interaction
- Programmers, designers, and users (with training) can understand the diagrams
- Provides an unambiguous description for a small part of the world
- Easily translated to code

FSMs and UIs (cont)

- Needs to be complemented by non-formal language to be useful
  - “When in the disabled state, the user can’t interact with the button at all. It’s enabled and disabled programmatically.”
  - Definitions could include language like “Change the appearance of the button in the disabled state to a softer, lighter, less ‘present’ appearance.”
- Result is a semi-formal description

Consider



Font

Latin text font: +Body

Font style: Regular

Size: 32

All text

Font color

Underline style: (none)

Underline color

Effects

□ Strikethrough

□ Double Strikethrough

□ Superscript Offset: 0%

□ Subscript

□ Small Caps

□ All Caps

□ Equalize Character Height

Problems with FSMs

- Scale: Interesting UIs too complex to draw and understand
  - Don’t fit on a single sheet of paper; May be non-planar
- Inadequate expressive power
  - Too much need for duplication; can’t represent parallelism
    - Example: need to enter eight pieces of information, in any order, but each one only once
  - Anything interesting will have exceptions, leading to many states
  - Eg: hard to handle time
- Still useful for interfaces that must be bullet-proof; fragments of user interfaces

Desirable Properties

- User Interfaces…
  - have lots of affordances (hopefully relatively modeless)
  - have lots of factorable (decomposable) state
  - change in small (observable) increments

- If we had a generalization of state machines…
  - that allows many state machines at once
  - that all work in parrallel
  - that have small effects on one another

Propositional Production Systems

- A general computational system
- Emil Post proved equivalent to a Turing Machine in 1930’s
- Can be used to specify interactive behavior
- Used to describe the human brain, then used in AI

- Benefits:
  -
  -

PPSs: Definitions & Productions

- **Definitions**: formal symbols with natural language (non-formal) descriptions
- Defines the things that can appear in a production:
  - State variables
  - Input Events
  - Event Modifiers
  - Queries
  - Actions
- **Productions**: formal sets of conditions and actions
- <conditions> → <actions>
  - Events, Event modifiers, and Queries appear on the left side
  - Actions appear on the right
  - State variables on either side: Left => query state, right => set state to X.
  - All rules are evaluated on each event
  - All rules where all of the conditions are true have their actions evaluated in parallel

PPS Elements

- 5 different field types
  - Input Events
    - *mouseDown, *mouseUp, *mouseMoved
  - Input Modifiers
    - leftMseDown*, leftMseUp*, ctrlDown*, ctrlUp*
  - State Information
    - sliderActive, sliderInactive
  - Query Fields
    - ?validPasswd, ?stepUp, ?slider
  - Actions
    - !drawButtonUp, !dragStart

PPS Elements (2)

- Always one input event field
  - Input events are processed one-by-one so only one possible input event at a time
- Input Modifiers
  - One per modifier you will check
  - E.g. if you check shift, ctrl, and mouse button, you need 3 input modifier fields
  - Within a field mutually exclusive, but can combine input modifiers simultaneously
    - Each modifier independent

PPS Elements (3)

- State fields
  - Handle control issues
  - Encode current status of dialog (e.g. buttonDown, slider)
  - Typically one state field (one response to on-going user action)
- Query fields
  - Allow testing of conditions
- Action fields
  - Basically the things we want code to do
  - Should only appear on RHS.

Productions

- Productions are statements with 2 components
    - LHS = antecedant
    - RHS = consequent
- If LHS is all true, then production fires
    - If multiple LHS true, all fire in parallel (for theoretical convenience)

Production (2)

- Example:

***mouseUp, shiftDown*, selectClick -> !addSelect, selectModeIdle**
***mouseUp, shiftUp*, selectClick -> !newSelect, selectModeIdle**

- Meaning (1)
  - On mouseUp event, with shift down, in selectClick mode, then
  - Add to selections, switch to selectModeIdle
- Meaning (2)
  - On mouseUp event, with shift up, in selectClick mode, then
  - Unselect old and create a new selection, switch to selectModeIdle

Productions (2.5)

- Note
  - Input events, input modifiers, query conditions can only occur on LHS
    - These are things user does and information about those actions
    - Program has no control over these things
  - Actions can only occur on RHS
    - These are things that the program should do if LHS holds
    - Program does these things in response to characteristics of user input
    - These represent changes to model (or view)

Productions (3)

- Consider these two rules

***mouseUp, shiftDown*, selectClick -> !addSelect, selectModeIdle**
***mouseUp, shiftUp*, selectClick -> !newSelect, selectModeIdle**

- mouseDown productions

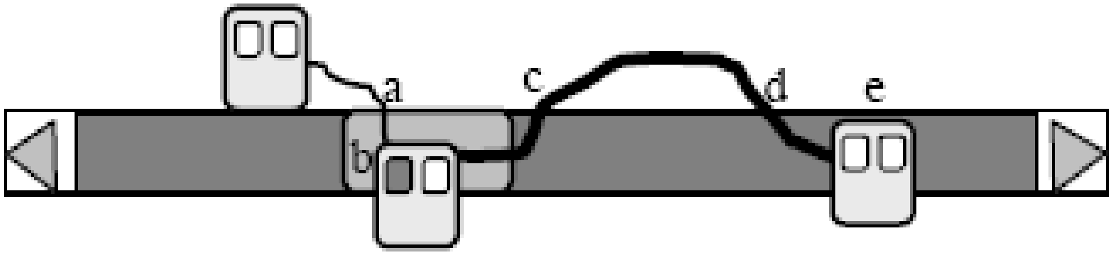
***mouseDown, ?onObject, selectModeIdle -> selectClick**
***mouseDown, ~?onObject, selectModeIdle -> clearSelect**
***mouseDown, drawModeIdle -> !newStroke, drawingGesture**

- Basically
  - On mouseDown, check to see if there's an object there.
  - If so, then on mouseUp add to selections
  - If not, then on mouseUp …

Scrollbar Example

- Input
  - {*mouseDown, *mouseMove, *mouseUp, *mouseExit }
- State
  - { idle, steppingLeft, steppingRight, pagingLeft, pagingRight, dragging }
- EssentialGeometry
  - { ?leftArrow, ?rightArrow, ?leftBody, ?rightBody, ?slider }
- Model
  - { !stepLeft, !stepRight, !pageLeft, !pageRight, !dragStart, !dragEnd, !dragScroll }
- Feedback
  - { !sliderActive, !leftArrowActive, !rightArrowActive, !bodyActive, !allPassive }

  –



Basic Productions

- **Basic stepping behaviour**
  - ***mouseDown, idle, ?leftArrow -> steppingLeft, !leftArrowActive**
  - ***mouseUp, steppingLeft -> idle, !allPassive, !stepLeft**
  - ***mouseDown, idle, ?rightArrow -> steppingRight, !rightArrowActive**
  - ***mouseUp, steppingRight -> idle, !allPassive, !stepRight**
  - ***mouseDown, idle, ?leftBody -> pagingLeft, !bodyActive**
  - ***mouseUp, pagingLeft -> idle, !allPassive, !pageLeft**
  - ***mouseDown, idle, ?rightBody -> pagingRight, !bodyActive**
  - ***mouseUp, pagingRight -> !allPassive, !pageRight**
- **Mouse sliding off control, e.g. stepper arrows**
  - ***mouseMove, steppingLeft, ~?leftArrow -> !allPassive, idle**
  - ***mouseMove, steppingRight, ~?rightArrow -> !allPassive, idle**
  - ***mouseMove, pagingLeft, ~?leftBody -> !allPassive, idle**
  - ***mouseMove, pagingRight, ~?rightBody -> !allPassive, idle**
  - ***mouseExit ~idle -> !allPassive idle**

Problems?

- Repeated behaviors
  - Only one !stepLeft event when *mouseDown on left arrow for example
- Add new productions to continue stepping

Basic Productions

- **Basic stepping behaviour**
  - ***mouseDown, idle, ?leftArrow -> steppingLeft, !stepLeft**
  - **steppingLeft, *mouseDown -> steppingLeft, !stepLeft**
  - ***mouseUp, steppingLeft -> idle, !allPassive,**
  - ***mouseDown, idle, ?rightArrow -> steppingRight, !stepRight**
  - **steppingRight, *mouseDown-> steppingRight, !stepRight**
  - ***mouseUp, steppingRight -> idle, !allPassive,**
  - ***mouseDown, idle, ?leftBody -> pagingLeft, !pageLeft**
  - **pagingLeft, *mouseDown -> !pageLeft**
  - ***mouseUp, pagingLeft -> idle, !allPassive**
  - ***mouseDown, idle, ?rightBody -> pagingRight, !pageRight**
  - **pagingRight, *mouseDown -> !pageRight**
  - ***mouseUp, pagingRight -> idle, !allPassive**
- **Mouse sliding off control, e.g. stepper arrows**
  - ***mouseMove, steppingLeft, ~?leftArrow -> !allPassive, idle**
  - ***mouseMove, steppingRight, ~?rightArrow -> !allPassive, idle**
  - ***mouseMove, pagingLeft, ~?leftBody -> !allPassive, idle**
  - ***mouseMove, pagingRight, ~?rightBody -> !allPassive, idle**
  - ***mouseExit ~idle -> !allPassive idle**

Special productions

- What if mouse slides off control while dragging slider
  - Adds a getFocus and releaseFocus action in a Focus action field
    - Focus {!getMouseFocus, !releaseMouseFocus }
  - Also new productions to handle this
    - *mouseDown, idle, ?slider -> !sliderActive, dragging, !dragStart(mousePoint), !getMouseFocus
    - *mouseMove, dragging -> !dragScroll(mousePoint)
    - *mouseUp, dragging -> !dragEnd(mousePoint), idle, !releaseMouseFocus

Propositional Productions Summary

- Encode state space in a set of fields
- Fields serve five purposes
  - Input events, input modifiers, state information, query fields and action fields
- After we have behaviour specified, realize in code
  - Coding can be long, but productions guide process
  - Each input event should be mapped to modification of state information and/or actions based on
    - Input event, state information, input modifiers, and query fields
- Best use
  - Custom control design, ensuring everyone is on same page with behavior
- Used in practice?

Advantages of PPSs

- No drawing! Do it all with a simple text editor. Compact.
- Fits UI needs well:
  - Formalize the model
  - Informal descriptions of interaction with the real world
  - Semi-formal descriptions of view and controller
- How does the complexity of a production system compare to the complexity of the actual code?
- How does this complexity compare to the alternatives?
  - Natural langage?
  - Scenarios/Mouse Event Diagrams?
  - FSMs?

Translating PPS into Code

- Order the conditions of each rule:
  - events, state variables (most used to least used), queries
- Sort the rules
- Group rules into applicable event listeners
- Translate the conditions
- If more than one rule has the same conditions, it probably means an error in the specification.

- (Textbook has more detail.)

Summary

- We use widgets to carry on a “dialogue” with the computer.
- We can use observations about human-human dialogues to help design human-computer dialogues.
- Typical human-computer dialogues have five parts (from the computer’s perspective): prompt, echo, accept trigger, acknowledge, respond
- Documenting these dialogues requires a semi-formal language because we are trying to bridge the gap between “open” users and “closed” computers.
- Scenarios, Finite State Machines, and especially Propositional Production Systems are useful tools for documenting human-computer dialogues.