CxStates

Little Smalltalk Exercise:
A dynamically defined state model
not based on the state pattern

presented by
Alfred Wullschleger
Swiss National Bank
The Author

- Smalltalker since 1992
- Project OVID at Fides Informatik (1992-1999)
  - OVID currently in production more than 11 years
- Project OASE at Swiss National Bank (since 1999 in production)
  - Financial Statistics from Swiss Banks and Companies
  - based on Gemstone/S and VisualWorks
  - ongoing development under full production
Motivation

• We needed a user configurable state model
  – so, preferably should not be class based
  – should allow easy communication with the „outer system“ = the system components which are not part of the state model itself

• Solution: CxStates
  – ANSI Event Model as implementation pattern
Basic classes (1)

- **CxBaseState**
  - defines a state
    - has a name
    - has a transitionTable
      - contains legal transitions to other states through transition entries

- **CxTransitionEntry**
  - defines a transition by a symbol and the new state
    - includes transitionActions
Basic Classes (2)

- **CxActionSequence and CxAction**
  - for execution of methods in the „outer System“
  - extensible analogous to the event model

- **CxStateEnsemble**
  - Defines a complete state diagram
  - Defines all legal transition symbols
Action execution

PreTransition Actions

Transition

PostTransition Actions

current State

new State

preTransitionActions in context of current state
postTransitionActions in context of new state
Action

A CxAction contains a message send. CxAction is declared in the transitionEntry. The message receiver is typically inserted during the transition.
Declaration of actions

• in CxBaseState:
  – whenCxPreTransition: aTransitionSymbol send: aSelector to: aReceiver
    • adds a CxAction to the PreTransitionActionSequence
    • aReceiver can be nil: the receiver can be dynamically set when the transition is executed
    • polymorphically delegated to CxTransitionEntry
  – whenCxPostTransition: aTransitionSymbol send: aSelector to: aReceiver
    • the same in context of the new state
CxTransitionContext

- Is an argument holder for the messages sent to the „outer system“
  - combines receiver and arguments for the declared action
  - includes currentState (either the pre- or the postTransition state!)

- CxBaseState>>doTransition:
  aCxTransitionContext
  - Basic method for the execution of the transition
CxBaseState>>doTransition:

• Executes the defined CxActions
  – By delegation to CxTransitionEntry>>doTransition:, and finally to CxAction>>doTransition:

• Returns
  – nil, when the transition is illegal, state unchanged
  – false, when the transition is inhibited, state unchanged
  – the new state, in all other cases

• Inhibit by a special CxAction: CxCondition
  – Must returns true or false
Demonstration

• Automatic Coffee Machine
  – Power switch
  – „Make Coffee“ button
  – grinding automatic (timer)
  – Heating (temperature sensor, not simulated)
  – brewing automatic (timer)
  – „Stopp brewing“ button
  – Water supply not simulated
Coffee States

Init

Switched Off

Switched On

Grinding/Heating

Heating

Brewing

PressButtonForCoffee

StopGrinding

cupFilled

StartBrewing

cancelBrewing

initializeMachine

switchPowerOn
Extensions

- **CxCondition**
  - allows to inhibit preTransitions by outer system conditions
  - analogous to CxActions but must return a boolean

- **CxPostTransitionEntry**
  - technical class for post states which do not contain transitionEntries for the corresponding transition symbol
  - PostTransitionEntries must be defined in the post state!
Thank you for listening

Questions?