UML 2.0 – Composite Structures and State Machines

Version 050212

Fois by Ø. Haugen and B. Møller-Pedersen
Overview of the lecture

- A very small repetition
  - class diagrams, sequence diagrams, use cases
  - what do they convey?
- Composite structures
  - of classes and collaborations
  - ports
- State Machines (basics)
  - what is an FSM – a Finite State Machine?
  - example of a finite state machine
- State Machines (advanced)
  - encapsulation (exit and entry points)
  - composite states and submachines states
Example - ATM

- Domain statement
  - An Automatic Teller Machine (ATM) is a system with mechanical as well as electronic parts. Its purpose is to provide a bank user with cash provided that the user can authenticate herself and she has adequate funds in her bank account.
  - She authenticates herself by presenting a card to the ATM cardreader, and a personal identification number (PIN) through the ATM keyboard.
  - The ATM is connected electronically and possibly through some kind of network to the bank such that the account status may be checked online.
  - The ATM is refilled with cash notes regularly or when the number of specific notes is below some limit.
  - The ATM may also provide foreign currency to the customer.
Class Diagram (1)

- **User**
- **ATM**
- **Bank**
- **Card**
- **Account**

** concepts – potential objects **

** associations – between concepts **

- User: * * 1
- ATM: * 1
- Bank: 1
- Card: 1
- Account: 1

myAccounts
Class Diagram (2)

... but a given keyboard of a given ATM, can only be contained in that ATM

composition – between concepts

there may be keyboards in other things than ATMs

CardReader  Keyboard  Screen  CashDispenser
Use Case Diagram

- **User**
- **Bank**
- **CashRepository**
- **Withdrawal**
- **Authentication**
- **Currency**

**ATM**

- **use case – is formally a class, not a behavior**
- **actor – outside the subject**
- **the subject – the system to be made**
Interaction Diagram

- continuation
- interaction use
- combined fragment

lifeline – typed by a class – referring to a part/property
Composite Structure – Collaboration

What is inside one such ATM object?
Composite structure – class

ATM

CardReader
Keyboard
Screen
CashDispenser

User-Reader
User-Keyboard
User-Screen
User-Cash

ATM-bank

part
port
connector
Finite State Machines

- **Finite**
  - a finite number of states
  - [here] a *small* number of *named* states

- **State**
  - a stable situation where the process awaits stimuli
  - a state in a state machine represents the history of the execution

- **Machine**
  - that only a stimulus (signal, message) triggers behavior
  - the behavior consists of executing transitions
  - may also have local data
EnterPIN state machine

<<statemachine>>
EnterPIN

n:integer
PIN: integer

initial (pseudo) state

local data

start transition

transition – [guard] trigger / action

[\n<4\]digit/

n++;

PIN= PIN+digit*10^{3-n}

send(msg("Give PIN")); n=1; PIN=0

send(Code(cid,PIN))

waitOK

enterDigit

[n<4]digit/n++; PIN= PIN+digit*10^{3-n}

ok

nok

nok

ok

ok

start transition

transition – [guard] trigger / action

[\n<4\]digit/

n++;

PIN= PIN+digit*10^{3-n}

send(Code(cid,PIN))

waitOK

enterDigit

[n<4]digit/n++; PIN= PIN+digit*10^{3-n}

ok

nok

nok

ok

ok

state – simple

exit point – definition
Statemachine for the ATM

sm ATM

Idle

/:EnterPIN

/:Service

/:Withdrawal

/:Status

CardOut

cardTaken

exit point – usage

submachinestate – usage

simple state with entry clause

Idle

/authN=0

CardId(cid)

/:EnterPIN

/nok

[authN==3]/
authN=0
send(msg("illegal entry"));

[authN<3]/
authN++;
send(msg("Try again"));

ok

/:Service

Withdrawal

status

cancelled

/:Withdrawal

/:Status

ok

entry: send(card)
Attributes of the ATM

- Statemachine is a Classifier (that is class-like):
  - Attributes
  - Operations (local actions)

- authN: number of tries
- cid: card id
- sa: selected amount
- aa: available amount

```
<<statemachine>>
ATM

authN: integer
cid: integer
sa: Amount
aa: Amount

sendMoney(a:Amount)
```
State machine Withdrawal

sm Withdrawal

:GetAmount

cancelled

send(CheckAccount(sa))

cancelled

again

nok/
send(msg("Amount too large"))

VerifyTransaction

ok/
sendMoney(sa);
send(Receipt(sa));

ok

entry point – usage
Simple GetAmount

```
sm GetAmount

Send(msg("select amount"))

:SelectAmount

Send(msg("select another amount"))

amount(sa);

cancel

cancelled

again

final state – definition

entry point – definition
```
And we repeat – The Knoble game
The game

- A game administrator controls the game
- Invites the players
- The players make a draw like:

- The game administrator calculates the scores
Composite Structure: Knoble Game
A simple State Machine

Initial State

Simple states

Transitions:
trigger / action or trigger ^send
Motivating Composite State / Substate machine

- Some transitions are more global than others
  - When a television set is ON you may be in textual mode or in normal television mode
  - When in textual mode, the digits have a different interpretation than they do in normal television mode
  - When you turn OFF the set, it goes black regardless of whether it is in textual or television mode

- One achieves
  - better overview
  - more compact code, transitions need in lesser degree be duplicated
  - potential reuse of composite states
  - composite states are potential units for on-line replacement
Remaking the Knoble player state space

- **Connected**
  - Chat-transition is equivalent for these four states

- **Disconnected**
  - disconnect-transition is equivalent from four states

- **Connecting**

- **NoGame**
  - connected

- **Signing**
  - connected

- **SignedOn**
  - connected

- **Playing**
  - connected
Substate machine: Connected

- Connected
- NoGame
  - Scissors|Paper|Stone
  - responseTimer2 / PlayerSignal
  - cancelPlay / stopTimer1&2
- Signing
  - responseTimer1 / startTimer2
- SignedOn
  - startPlay / startTimer1
- Playing

Chat

Initiated in all states within the composite state, returning to History pseudo-state

default starting point of Connected

composite state – just a "state parenthesis"
Main state space revisited

one instantiation of the Connected submachine state
Another potential use of submachine states

- A state machine is well encapsulated
  - entry and exit points
  - its own data space
  - its own state space

- A submachine state represents the usage of such a state machine
  - it is content as long as the interface behavior is consistent with the encloser of the submachine state

- Can we change the definition of the state machine of the submachine state?
  - at compile-time (modeling-time)
  - at runtime?
Changing the behavior during runtime (before)

![Diagram showing the state transitions and actions for a player in different states such as noGame, signing, signedOn, and playing. The diagram includes transitions for Timeout1, Timeout2, SignOn, StartPlay, and CancelPlay.]
Changing behavior during runtime (after)

When the Player state machine enters the initial state of Connected it is transferred to the new state definition.

Before behavior change

After behavior change

Player

Connected

Connected /*new*/