32536:
Object Oriented Modelling

Vending Machine

Terry Corlet,
Josip Lozina and
Stephan Brumme

June 3rd, 2004

Agenda

1. Introduction
2. Process
3. Implementation
4. UML Critique
5. Conclusion

Introduction - I

- identifying the objective

Introduction - II

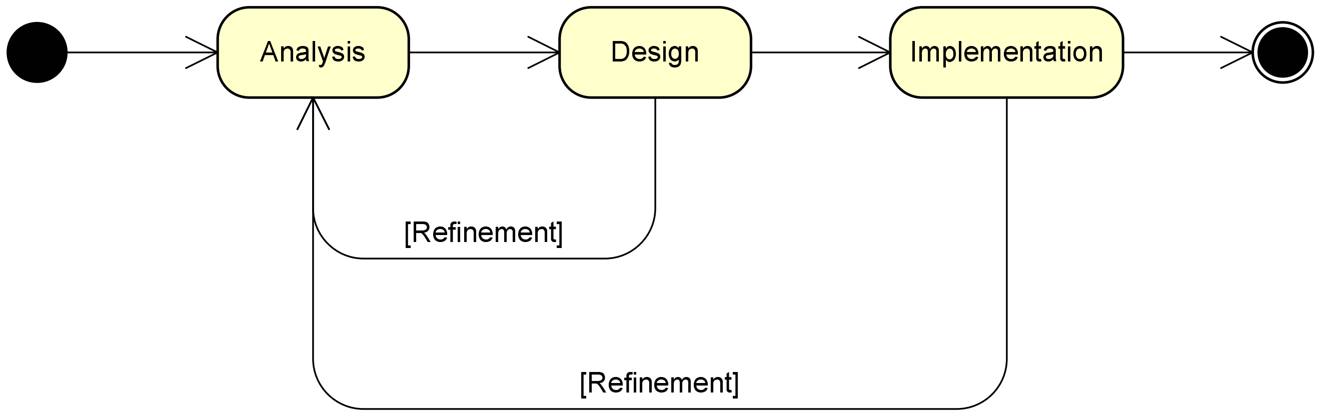
- setting up a plan
  - who ?
  - when ?
  - where ?
  - what ?

- decide on notation
  - UML

- team member roles
  → certain experiences ?

Process - I

- iterative approach
  - start with a simple model
    → refine to build the final model



  - we came up with 8 revisions !

Process - II

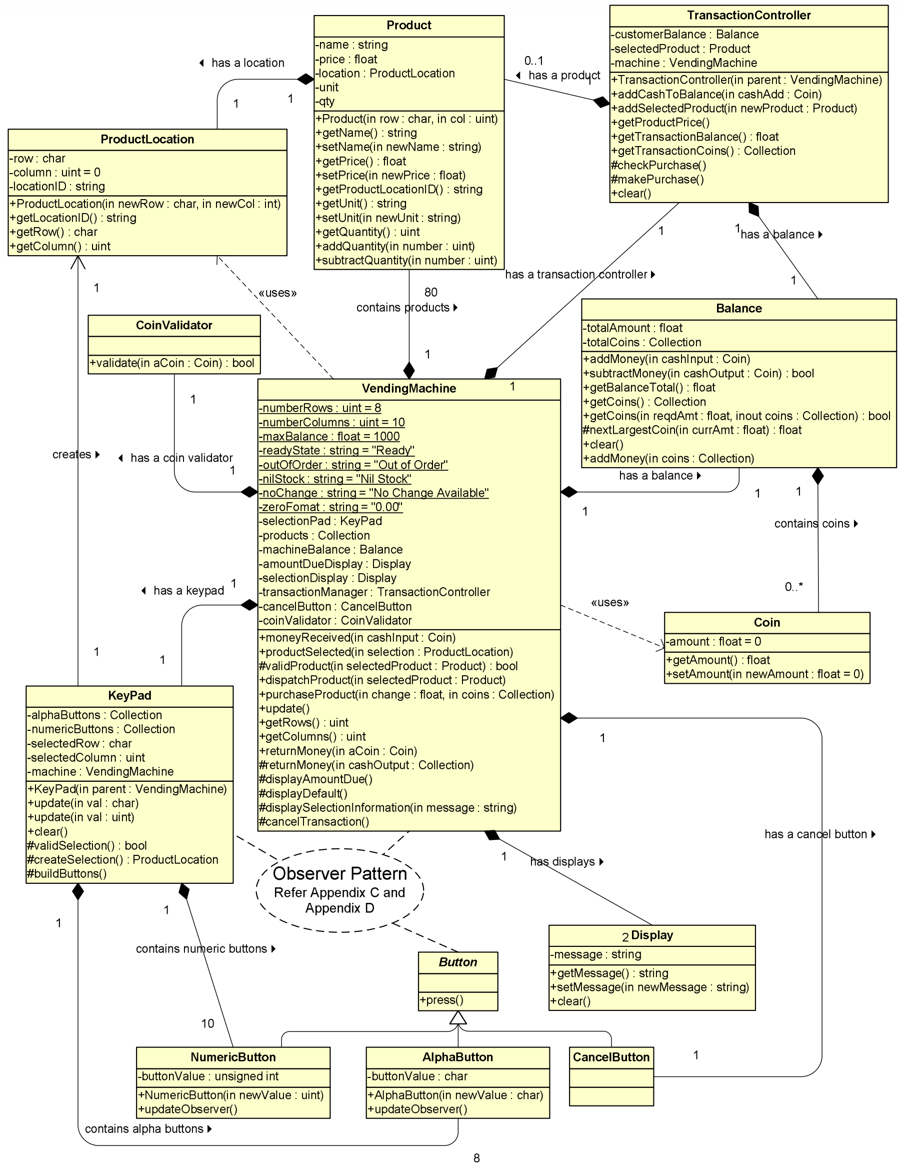
- identifying requirements
  - what is clear ?
    - 80 items available
    - several buttons
    - and many more …
  - what is ambiguous ?
    - which currency ?
      - 5 cent rounding
      - smallest accepted coin
    - items out of stock
    - tracking sales
    - and many more …

!

?

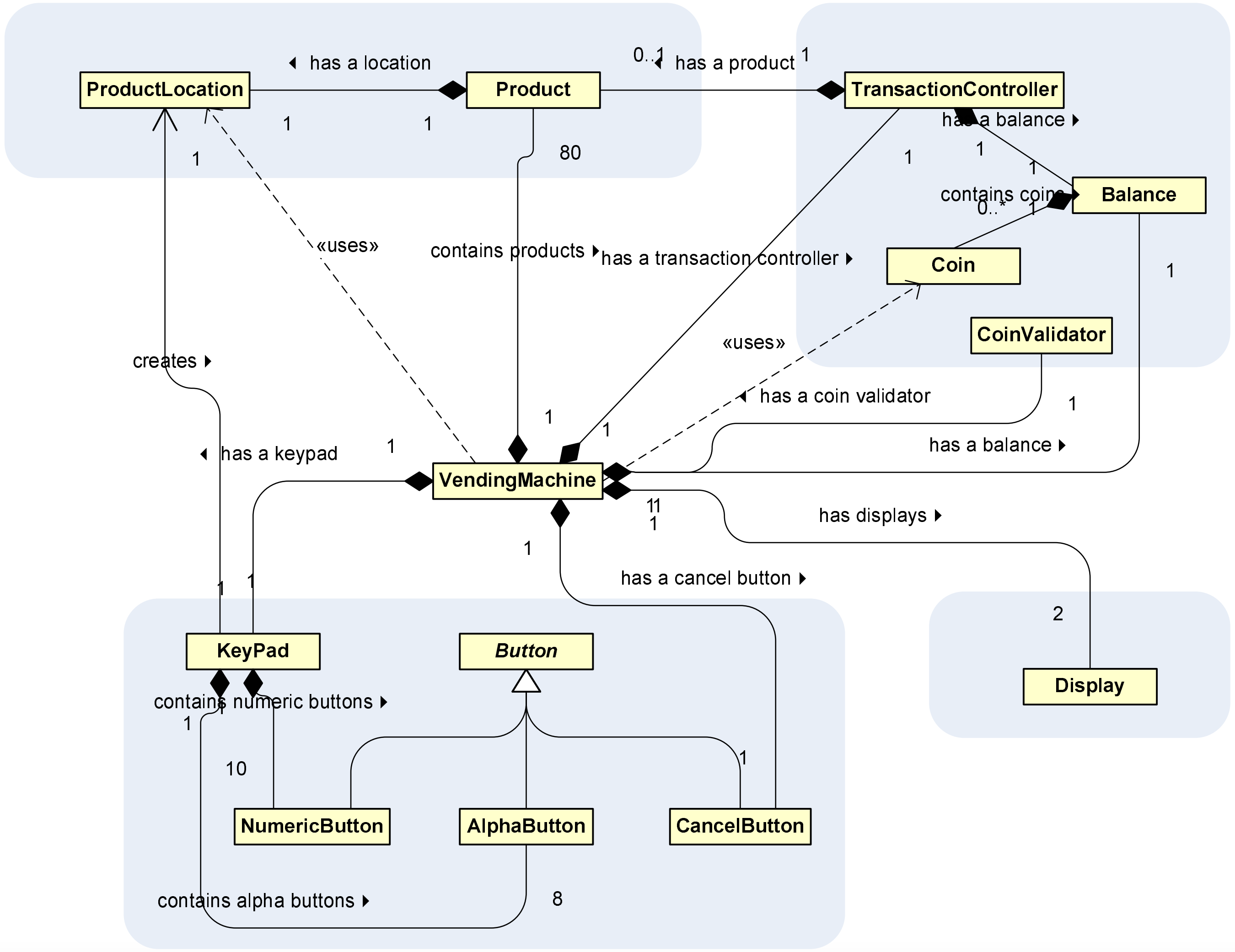
Process - III

- CRC modelling
  - class diagram is integral
  - look for adequate names
  - semantics
- identifying patterns
  - well-known techniques
  - often directly mapped to UML structures


Product
-name : string
-price : float
-location : ProductLocation
-unit
-qty
+Product(in row : char, in col : uint)
+getName() : string
+setName(in newName : string)
+getPrice() : float
+setPrice(in newPrice : float)
+getProductLocationID() : string
+getUnit() : string
+setUnit(in newUnit : string)
+getQuantity() : uint
+addQuantity(in number : uint)
+subtractQuantity(in number : uint)
TransactionController
-customerBalance : Balance
-selectedProduct : Product
-machine : VendingMachine
+TransactionController(in parent : VendingMachine)
+addCashToBalance(in cashAdd : Coin)
+addSelectedProduct(in newProduct : Product)
+getProductPrice()
+getTransactionBalance() : float
+getTransactionCoins() : Collection
#checkPurchase()
#makePurchase()
+clear()
ProductLocation
-row : char
-column : uint = 0
-locationID : string
+ProductLocation(in newRow : char, in newCol : int)
+getLocationID() : string
+getRow() : char
+getColumn() : uint
has a location
has a product
0..1
1
1
has a transaction controller
has a balance
1
1
1
«uses»
80
contains products
1
CoinValidator
+validate(in aCoin : Coin) : bool
Balance
-totalAmount : float
-totalCoins : Collection
+addMoney(in cashInput : Coin)
+subtractMoney(in cashOutput : Coin) : bool
+getBalanceTotal() : float
+getCoins() : Collection
+getCoins(in reqdAmt : float, inout coins : Collection) : bool
#nextLargestCoin(in currAmt : float) : float
+clear()
+addMoney(in coins : Collection)
VendingMachine
-numberRows : uint = 8
-numberColumns : uint = 10
-maxBalance : float = 1000
-readyState : string = "Ready"
-outOfOrder : string = "Out of Order"
-nilStock : string = "Nil Stock"
-noChange : string = "No Change Available"
-zeroFomat : string = "0.00"
-selectionPad : KeyPad
-products : Collection
-machineBalance : Balance
-amountDueDisplay : Display
-selectionDisplay : Display
-transactionManager : TransactionController
-cancelButton : CancelButton
-coinValidator : CoinValidator
+moneyReceived(in cashInput : Coin)
+productSelected(in selection : ProductLocation)
#validProduct(in selectedProduct : Product) : bool
+dispatchProduct(in selectedProduct : Product)
+purchaseProduct(in change : float, in coins : Collection)
+update()
+getRows() : uint
+getColumns() : uint
+returnMoney(in aCoin : Coin)
#returnMoney(in cashOutput : Collection)
#displayAmountDue()
#displayDefault()
#displaySelectionInformation(in message : string)
#cancelTransaction()
creates
has a coin validator
has a balance
contains coins
has a keypad
«uses»
0..*
Coin
-amount : float = 0
+getAmount() : float
+setAmount(in newAmount : float = 0)
KeyPad
-alphaButtons : Collection
-numericButtons : Collection
-selectedRow : char
-selectedColumn : uint
-machine : VendingMachine
+KeyPad(in parent : VendingMachine)
+update(in val : char)
+update(in val : uint)
+clear()
#validSelection() : bool
#createSelection() : ProductLocation
#buildButtons()
has a cancel button
Observer Pattern
Refer Appendix C and Appendix D
has displays
2
Display
-message : string
+getMessage() : string
+setMessage(in newMessage : string)
+clear()
contains numeric buttons
Button
+press()
10
NumericButton
-buttonValue : unsigned int
+NumericButton(in newValue : uint)
+updateObserver()
AlphaButton
-buttonValue : char
+AlphaButton(in newValue : char)
+updateObserver()
CancelButton
contains alpha buttons
8

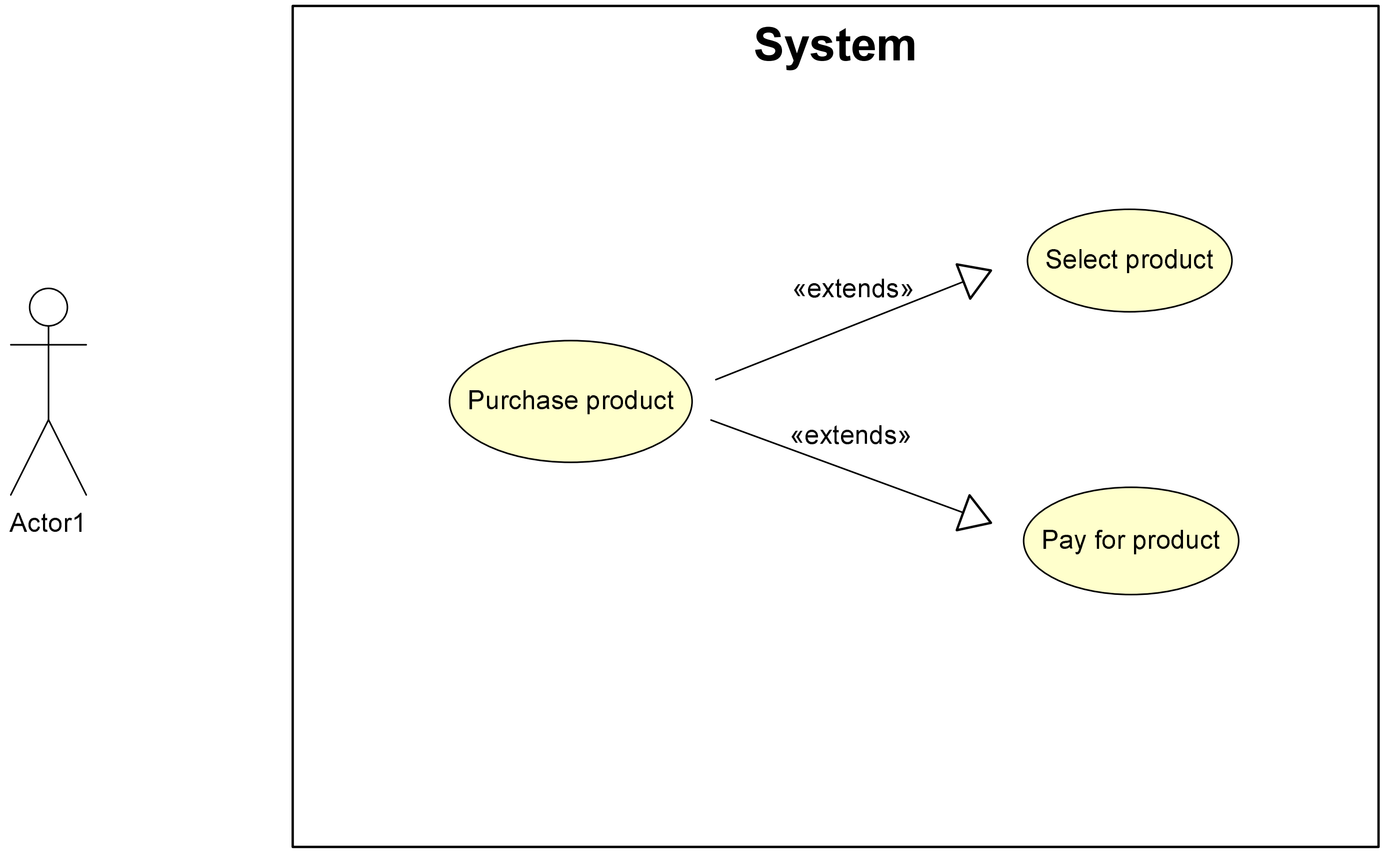
Process - IV

- Class diagram



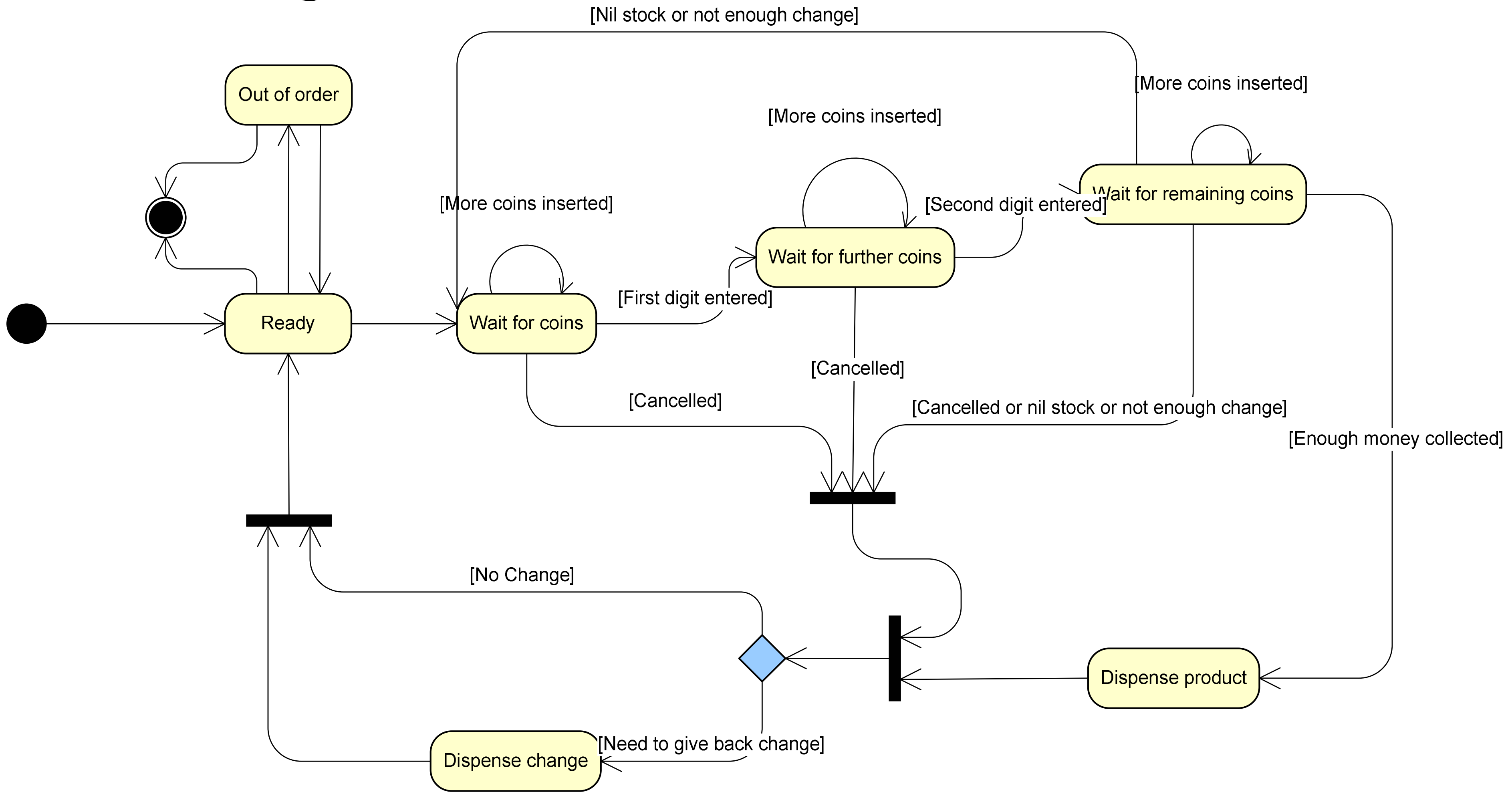
Process - V

- Use case diagram



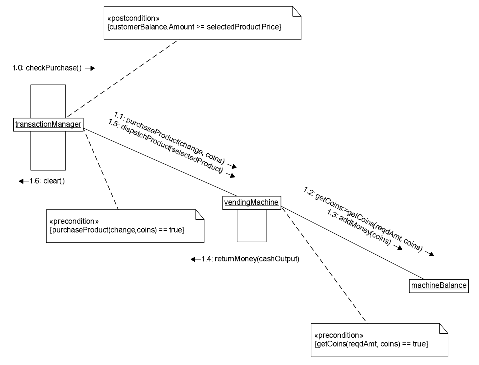
Process - VI

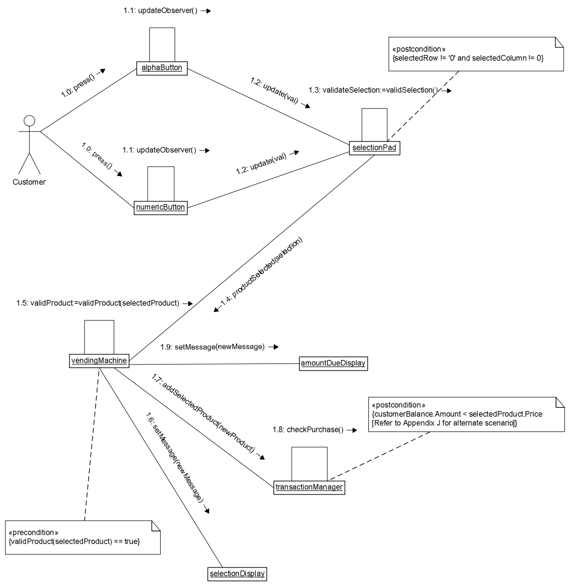
- State diagram

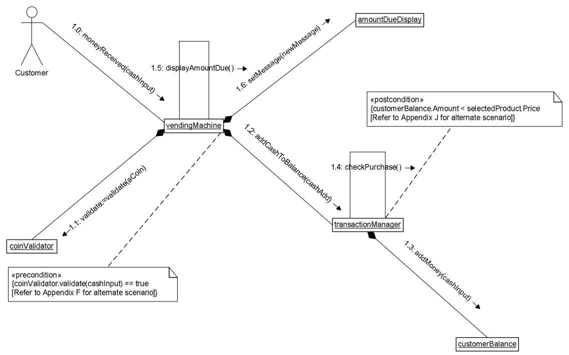


Process - VI

- Collaboration diagrams
  - mapping state diagram to OO structure
  - several diagrams !

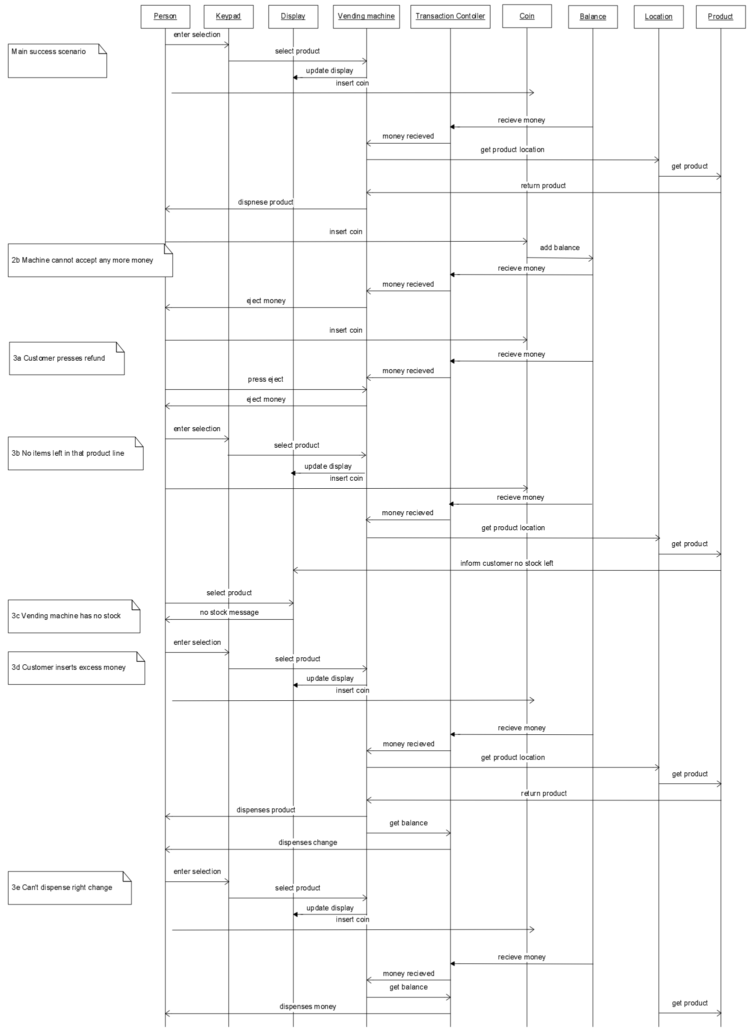

«postcondition»
{customerBalance.Amount >= selectedProduct.Price}
1.0: checkPurchase()
transactionManager
1.6: clear()
1.1: purchaseProduct(change, coins)
1.5: dispatchProduct(selectedProduct)
«precondition»
{purchaseProduct(change,coins) == true}
vendingMachine
1.4: returnMoney(cashOutput)
1.2: getCoins:=getCoins(reqdAmt, coins)
1.3: addMoney(coins)
machineBalance
«precondition»
{getCoins(reqdAmt, coins) == true}


1.1: updateObserver()
alphaButton
1.0: press()
1.2: update(val)
«postcondition»
{selectedRow != '0' and selectedColumn != 0}
1.3: validateSelection:=validSelection()
Customer
1.0: press()
1.1: updateObserver()
1.2: update(val)
selectionPad
numericButton
1.4: productSelected(selection)
1.5: validProduct=validProduct(selectedProduct)
vendingMachine
1.9: setMessage(newMessage)
amountDueDisplay
1.7: addSelectedProduct(newProduct)
1.8: checkPurchase()
«postcondition»
{customerBalance.Amount < selectedProduct.Price
[Refer to Appendix J for alternate scenario]}
1.6: setMessage(newMessage)
transactionManager
«precondition»
{validProduct(selectedProduct) == true}
selectionDisplay


Customer
1.0: moneyReceived(cashInput)
1.5: displayAmountDue()
1.6: setMessage(newMessage)
amountDueDisplay
vendingMachine
«postcondition»
{customerBalance.Amount < selectedProduct.Price
[Refer to Appendix J for alternate scenario]}
1.2: addCashToBalance(cashAdd)
1.4: checkPurchase()
1.1: validate:=validate(aCoin)
coinValidator
transactionManager
1.3: addMoney(cashInput)
«precondition»
{coinValidator.validate(cashInput) == true
[Refer to Appendix F for alternate scenario]}
customerBalance

Process - VI

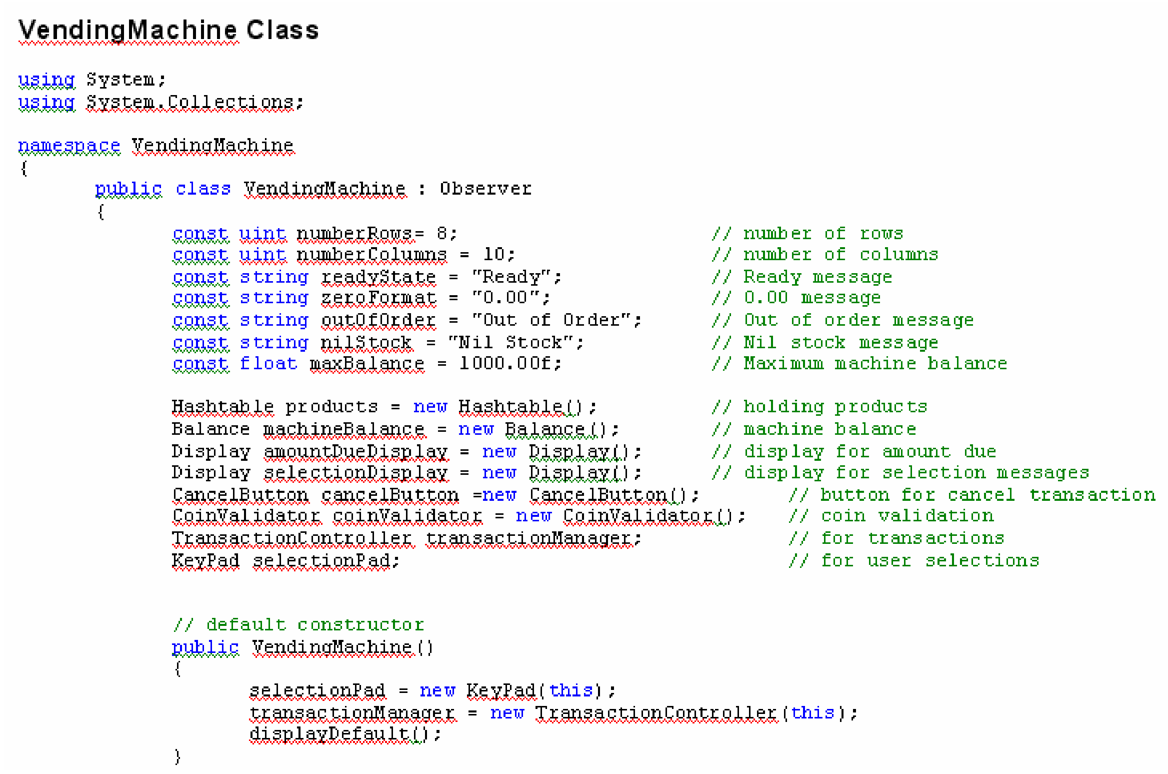
- Sequence diagram
  - emphasize temporal relationships



Implementation

- done in C#

- to test robustness of our design



VendingMachine Class

```
using System;
using System.Collections;

namespace VendingMachine
{
        public class VendingMachine : Observer
        {
                const uint numberRows= 8;                       // number of rows
                const uint numberColumns = 10;                  // number of columns
                const string readyState = "Ready";              // Ready message
                const string zeroFormat = "0.00";               // 0.00 message
                const string outOfOrder = "Out of Order";       // Out of order message
                const string nilStock = "Nil Stock";            // Nil stock message
                const float maxBalance = 1000.00f;              // Maximum machine balance

                Hashtable products = new Hashtable();           // holding products
                Balance machineBalance = new Balance();         // machine balance
                Display amountDueDisplay = new Display();       // display for amount due
                Display selectionDisplay = new Display();       // display for selection messages
                CancelButton cancelButton =new CancelButton();          // button for cancel transaction
                CoinValidator coinValidator = new CoinValidator();      // coin validation
                TransactionController transactionManager;               // for transactions
                KeyPad selectionPad;                                    // for user selections


                // default constructor
                public VendingMachine()
                {
                        selectionPad = new KeyPad(this);
                        transactionManager = new TransactionController(this);
                        displayDefault();
                }
```

UML Critique - I

- UML is not the customer's language !
  - UML is a mix of several notations
- too many structural details
  - and no consistent level of detail (use cases vs. class diagram)
- no unique algorithm to design UML diagrams
  - hundreds of possibilities to model a problem with UML
- hard to draw diagrams without software (Visio)

UML Critique - II

- UML violates basic rules of visualization
  - human perception
  - Bertin's variables

- no way to verify and validate requirements

→ try to take a look at competing modelling languages !

Conclusion

- all three team members have different background
  - … but UML helped to speak the same language !
- initial effort to learn UML
  - but the last meetings were quite efficient and effective

  → time initially spent paid off
- software tools seem to get better and better
- developer community accepts UML
  - now essential skill of advanced developers

Questions