CSSE 374 – Software Architecture and Design I

Homework 3

Objective

To apply what you have learned about UML Logical Architecture by:

1. Creating a preliminary Logical Architecture for the BrickBuster Video System (BBVS), and
2. Detailing selected portions of BBVS with Interaction Diagrams and Design Class Diagrams.

Due Date

5 p.m., Tuesday, Week 4, January 4th, 2011. **EXTRA CREDIT**: If you turn this homework in by 11:59pm on Friday, December 17th, 2010, you can receive 2 points (20%) extra credit on this assignment.

Tasks

1. Review the attached Domain Model, SSDs, Use Cases, OC's, and partial package diagram for BBVS. While these are not complete, they should supply the necessary information to complete the assignment.
2. Let's now consider the specific application of interest, in the BBVS domain, to be the one that sells videos to customers. (There would be others, like an application for provisioning the online inventory.) Develop a preliminary Logical Architecture that allocates the packages to appropriate layers and partitions for the customer application. Be sure to indicate dependencies between packages.
3. Choose two system operations from the provided SSDs. For each, draw an interaction diagram showing the detailed messages and objects involved in implementing the operation. One of your interaction diagrams must be a sequence diagram and one must be a communication diagram. Hint: These interaction diagrams likely represent some subset of the action within one of the provided SSDs.
4. Draw a design class diagram for the domain layer of BBVS following the guidelines in the book (Chapter 16) and discussed in class. This layer should be general enough to be useful for multiple BBVS applications. Our partial domain diagram, which has its own comments, can be used to get ideas, but we hope you put your own thoughts into the overall organization, and justify those in your comments to accompany your figure.

Per the suggestion in section 16.21 of the book, you should work on tasks 3 and 4 simultaneously. Also, recall that scans of neatly drawn pen and paper sketches are adequate for homework (though not for projects). There is a scanner in F217.

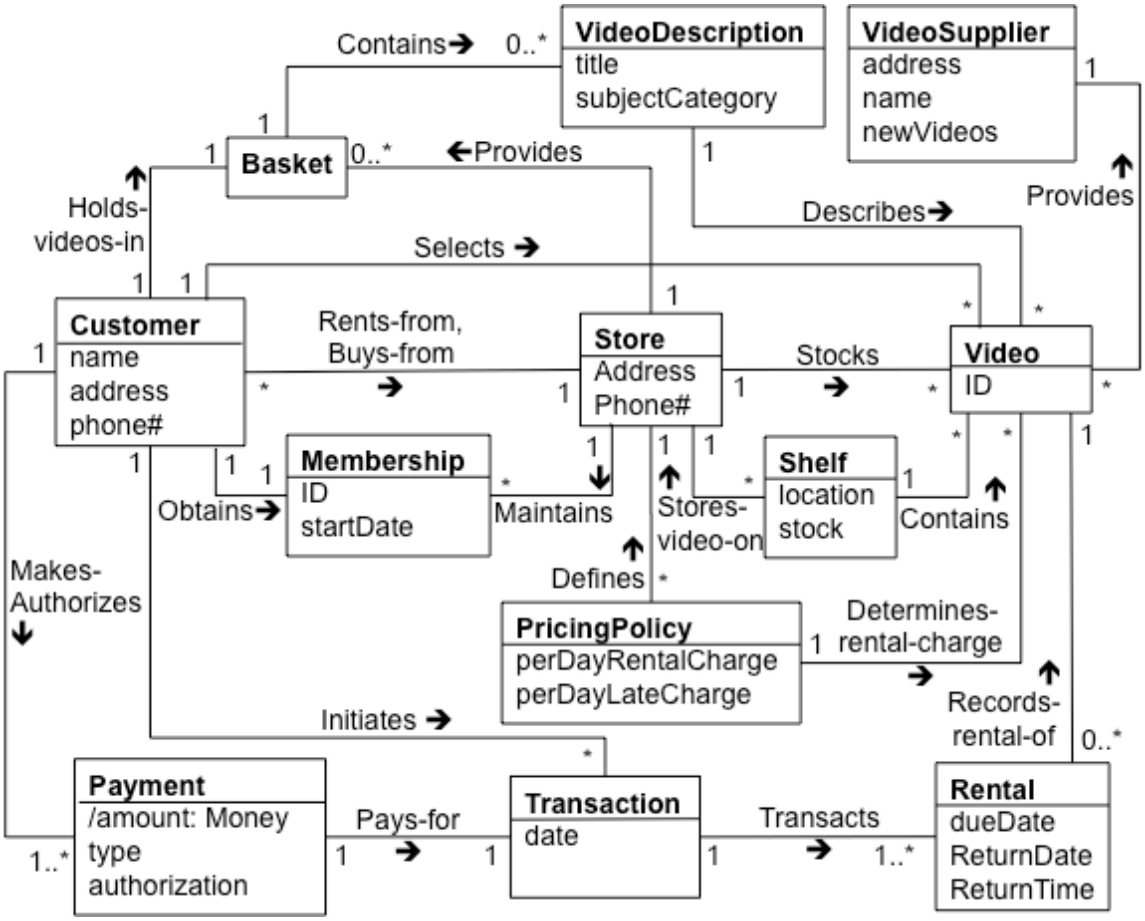
Submitting Your Work

Please submit your BBVS Logical Architecture as a single document as a single document to the appropriate Angel Homework dropbox.

Steve's Section: You can submit either a **pdf** file **or** a **Word** document. Name your document *HW3-Logical*, with the appropriate extension (*pdf*, *doc*, or *docx*). Please include your name on the first page of the document.

Shawn's Section: Please submit a **pdf** file with a cover page containing your Name, Assignment Title, Date, and Campus Mail number. Please name the document: <your last name>HW3-LogicalArch.pdf (e.g., Bohner-HW3-LogicalArch.pdf)

A Domain Model for BrickBuster Video System (Homework 1):

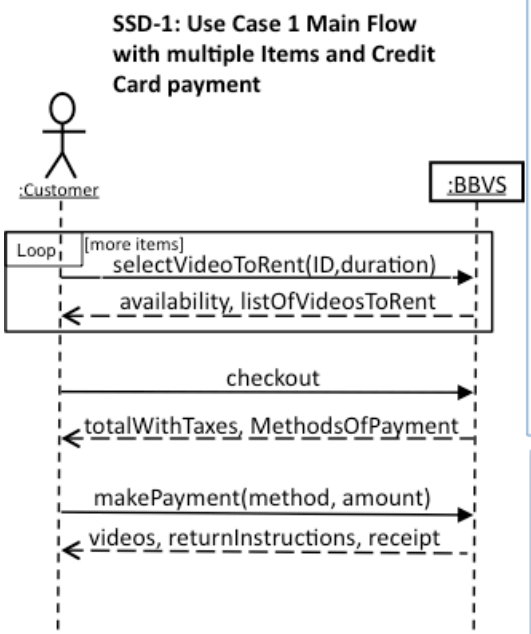


The Domain Model depicted above provides a basic framework from which the Interaction Diagrams, Design Class Diagrams, and Logical Architecture packages/components can be derived. In the figure above, the Customer is a key class that many of the other classes revolve around. The Customer Selects Videos, Buys-from or Rents-from the Store those Videos, and Initiates Transactions, including Making and Authorizing Payments.

New Videos are Provided by Video Suppliers while older releases are sold to customers. The Customer selects from Videos (in the store) from the stock on Shelves, and holds them a Basket – actually, like in a store, these just contain the empty case with the Video Description. The selected videos are then rented or purchased in the rental transaction. Hence, Transaction also plays a central role - it is Paid-for by a customer Payment, and it Transacts the Rental of the Videos. The Rental also is key - it Records-rental-of Videos. The Store is in multiple activities, in that it is where the Customer Rents-from, and it also Stocks the Videos.

System Sequence Diagrams and Operation Contracts

These SSDs and OCs represent key areas of the application of interest, within the domain.



Operation: selectVideoToRent(ID:VideoID, Duration:Days)
Cross-References:
- Use Case 1, SSD-1, Domain Model

Preconditions:
- Customer "Cust" is a member
- Cust item is in basket
- Cust item is in stock
- Cust is done selecting videos

Postconditions:
- Total with taxes was calculated and Payment Attribute amount was updated
- Payment was received and associated with Transaction
- Videos were packaged and associated with Transaction
- Cust item was associated with this sale by this customer
- Rental Transaction was Completed

Operation: makePayment(method: methods, amount: Money]
Cross-References:
- Use Case 1, SSD-1, Domain Model

Preconditions:
- Customer "Cust" has valid Credit card "CC"
- Cust has indicated he done selecting
- There is at least one item in Cust's sale

Postconditions:
- CC authorizer has Authorized Cust's Transaction
- Cust was given receipt and rental return instructions

From Use Case 1, we have the typical scenario where a customer buys multiple items, and the box around that part is to indicate this loop. The OC's detail two important parts of the transaction, the renting of individual items (videos), and the acceptance of payment, after which the customer can watch their videos.

Use Cases:

In getting these use cases form the customer, we tried to make them separate from the way the customer now does business. That is, we wanted them to be "how they wanted the new system to work." At the same time, we wanted to keep the use cases general enough that they might apply to more than one way of running a video business.

UC1: Customer rents videos

Preconditions: Customer has a membership, has selected videos they want to rent, and made the system aware of their choices.
Actor: Customer (if self-service or remote), or store associate (if in a store).

Main flow:

1. Actor indicates to rent first item (e.g., clicking "rent" on a networked device, or scanning it physically in a store).
2. System verifies immediate availability, waits for actor to make next option.
3. Actor indicates they are done selecting.
4. System shows total, prompts for payment.
5. Actor selects method of payment, entering additional data if needed (e.g., credit card number).
6. System verifies the payment has gone through, schedules the goods for rental (e.g., sets up a window to click on to view the video remotely, or tells the store clerk where to find the DVD).

Alternate flows (among many):

2a. System tells actor that the video is not currently available, and provides information on when it will be.
3a. Actor buys additional items, in the same way, if desired, returning to step 3 after each.
6a. System rejects method of payment, asks actor for alternative.

Postcondition: Rental transaction is complete.

UC2: System lets customer select videos to rent

Preconditions: None
Actor: Customer

Main flow:

1. "Store" shows videos available for rental. (This could be in stages, such as indicating areas of the store for customer to browse, or responding to a search request.)
2. Customer picks a video to rent.

3. System shows items in "basket," suggests checkout.

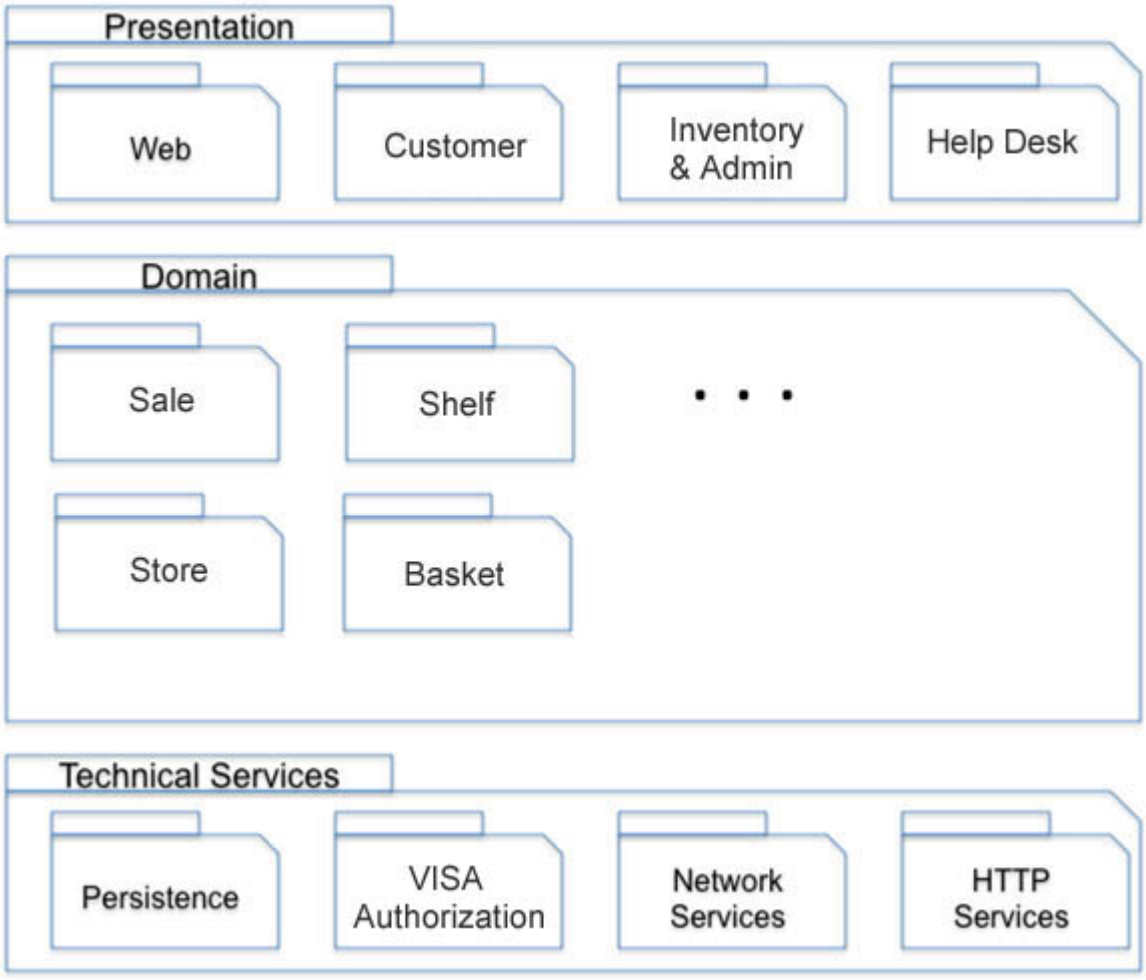
Alternate flows:

1a. Item is not in stock. "Store" shows when it is available, if possible.
2a. Customer continues in this loop, selecting multiple videos.

Postcondition: Customer has products ready to rent.

Partial Package Diagram (Logical Architecture)

This is the start of a package diagram, which, of course, you can diverge from as desired in your own depiction of layers, packages and classes. As with all figures, you should include a paragraph or two describing (and justifying your design choices).



In this example, we presumed that there would be a general web interface provided for web-based services. Perhaps "customer" should appear within that? However, we do have other ways of communicating with the customer,

say, by email. You may want to make your own choice as to how these parts of the presentation layer should be packaged!

In many customer service systems like this, the help desk has its own specialized interface for these representatives to be most productive. And people changing the inventory or doing other admin activities would need their own interface, too.

We only hinted at the domain layer! So, you should very much make this into something better. And, don't forget, you really have just a particular application within that domain, as your initial focus.

The technical services, or other layers supporting the domain and the applications, would similarly have additional components we didn't list.