CS 255 Systems Analysis and Design 5/29/2025

SNHU Clifton Brann

Module 4 Assignment

After examining the UML class diagram and considering the question what are the functions of the online storefront, the different classes in the diagram show the different functions of the store. In the strictest terms a class like User has a function of verifyLogin. But the class User represents the idea that there are going to be unique individual users for the system. These unique users will need to enter their own unique and private data into the system, so the class Customer will hold that data, and produce it when called to. The Shopping Cart functions as something that will hold the items selected by the customer, and the Order class actually places the order with the help of Order Details and Shipping Info. Order Details provides the total price of the items in the cart when the order is placed, and Shipping Info provides the most current address information given by the customer. Then there is the one class that is a little different, the Administrator class. It inherits most of its attributes from the User class, but it has the function of updating the catalog.

So to answer the question what are the functions of the online storefront, I would say that they are to break the storefront down into parts that are interdependent, but each do their own job. The User class functions to allow customers from the Customer class to login to the store and enter or maintain their demographic and payment information. The Shopping Cart functions to hold the Customer’s chosen items, until the Customer has committed to the purchase through the Order class, which will actually place the order and trigger the Order Details class to determine the total of the items purchased by the Customer, and apply the most updated shipping information to the order. The Administrator class has the ability to update the catalog to make sure all the items are current, and since it inherits a lot of its attribute from User, anyone using the Administrator class will have to log in through the User class.

And in this model the different functions of the system are represented by the different classes, each class will do a particular part of the operations in the online storefront. The class User will validate the logins from Customer and Administrator, which will allow them to proceed to the ‘shopping area’ with a cart that they can place items in for purchase or consideration. Once the items in the cart have been decided upon, the Order calls will be called and it will place the order, with input from Shipping Info and Order Details. So each of the parts of the storefront are represented by the different classes shown in the UML diagram.

There are two kinds of users in this example. The first is the Customer, who will hopefully be purchasing items from the online store. The there User is the Administrator, from the looks of the diagram, the Administrator is responsible to update the catalog, to keep the items listed for sale current, so that a Customer can’t order more of an item than is available, or an item that is not longer available.

Based on the UML diagram both the Customer and Administrator classes inherit most of their attributes from the User class. This is a good example of polymorphism, as the class user itself does not change, but the data it holds can be used by two other classes that have different uses and functions.

Depending on the class, each class will behave differently, the attributes in each class will hold the data as specified; as integers, strings, dates, or float values. As an example the Customer class will hold the data for a particular customer and so will inherit from User: the user id, password, login status and registration date. The Customer class itself will contain the name, address, email, shipping info, credit card info, and account balance data. The Customer class has the ability to update the profile information, as well as as register a new user and login the customer.

In my opinion the UML class diagram is complete. This diagram is lacking a payment processing class or package, along with any necessary supporting classes that would, for example, calculate shipping costs, calculate sales tax as appropriate, and allocate the tax to a separate account to be paid to the appropriate body at the appropriate moment. There is also at least two data base links missing. The stores inventory should be stored in catalog/inventory database so that the customers can search through the current items easily, with the most accurate counts available from frequent regular updates. There should also be a customer database to store all the customer data effectively, securely, and efficiently, instead of depending on the web store to store and maintain the data.

The amount of data the would need to be held in the storefront for both the store catalog with all the items, picture, description etc, and the customers demographic and payment information would quickly overwhelm the system if the site became very popular. If there were several thousand items, with the accompanying picture, descriptions, quantity on hand, and other data, would quickly cause a lot of lag if it were not managed in an efficient manner, through the use of a database. And if the catalog was included in the storefront, the software for the storefront would have to be recompiled every time the inventory would need to be updated, which would lead to significant downtime and inefficiency.

The black diamond shows the composition form of aggregation. The composition form is a strong form of aggregation, and it shows that the class that is on the black diamond end is the ‘whole’ class, and the class at the other end of the line is the ‘part of’ class. The whole class can stand alone, by itself. But the part of class is dependent on the whole class. In this example the Customer class is the ‘whole’ class, with Shopping Cart and Order being dependent on the Customer class. Without a Customer the can be no Shopping Cart or Order, but the reverse is true, the can be Customer without a Shipping Cart or an Order. And the Order class has further composition links, with Shipping Info and Order Details, which stands to reason, that without an Order there is no need for shipping, and if there is nothing being purchased then there is nothing to detail in Order Data’s, and nothing to add up, so there can be no account balance to pass to the Customer class.

It makes logical sense that Customers can exist without Orders, and that a Shopping Cart and and order are separate, just because an item is in the cart don’t mean it will get purchased. The different classes like Shopping Cart, Order, Order Details and Shipping info are related, but are only needed when a purchase is made, which cannot be done without a Customer (and possibly an Administrator but I’m not sure) .

The solid diamond is appropriate because it indicates a specific relationship between the classes. For the classes previously listed they all for the whole of a shopping experience, but each class represents an easier to use more manageable chunk of programming or coding. And I think the simpler the individual pieces are, the more reliable they are, because there is less that can go wrong with them. While it would be possible to code the shopping cart with all the functions of the other classes from the Order class, the Shipping Info class and the Order Details class, however it is more efficient to create more smaller and compact classes that do one thing very well, instead a single large class that does everything, but is prone to instability or is erratic because there are too many conflicting items in its attributes.

The process diagram breaks down purchasing system into the different steps, and shows how they relate to each other.The process diagram shows the inputs and outputs of each step. It clearly shows how each step of the purchasing system should function and what processes are related to those steps. It shows a good overall or general view of the system and its parts. However it does not show the internal nuts and bolts data and functions of the different parts that will actually make the system work. The process diagram is great for seeing the “what” in the system, as in “What is it supposed to do?”, but there is a lack of the “how”, like “How is it supposed to work?”. The process diagram answers, in broad strokes, the ‘what’ questions, without the level of detail required to answer the ‘how’ questions. There is also the lack of payment processing, databases for data storage, inventory system, and any mention of a returns or refunds handler. The missing payment processing step or system is important because it is firmly part of the system, and will require a certain amount of integration effort. At the same time, the payment system can be a parallel set of steps to the purchase system, or it can be fully integrated into the checkout module. From the reading, the client was expecting to use an existing solution like Square or Shopify as the payment gateway or processor, there should be some space or indication of where that would fit. This process diagram also lacks any indication of where databases would fit into the system; one would think that the company would want to store their customers data, keep records of purchases, store payment information, utilize aa catalog that accessed a database that was updated frequently for the most up to date inventory numbers, so customers can’t purchase more items then are in stock. There should also be an inventory system, to track what it available in the warehouse I real time, to avoid running out of product or running into backorder situations. There should also be a refunds and returns handler, to process and manage the returns from customers, wether it was mis-ordered product, unwanted product, or damaged product, and the refunds returns module should handle those situations.

The Object Model describes the how different parts of the system are related, and what they need to exist and what they will do when triggered. This model gives a much better idea of what the details of how the software will function after being written and tested. For example the User class has the attributes: userID, password, loginStatus, and registerDate. These attributes are the data inside the object, and the object will perform the function verifyLogin when called upon to act. The children of the User class are Customer and Administrator, and as such they will inherit the attributes from User, they will also contain the addition attributes of those objects, and perform other functions as listed in the lowest section of the rectangle. In my opinion the Object model answers a lot of the ‘how will the system function?’ questions. The level of detail about how data moves around and how the different modules of the system interact is much higher than in a process model. The different relationships between the different classes is also more detailed. The diagram is very specific in its notation and symbols, and this specificity shows the more details than just dependency and inheritance. It should the multiplicity of the relationship, and what kind of relationship exists between the different parts. There are still a missing parts: a payment processing module and databases with the online catalog/inventory, customer information, and order tracking/history. With the addition of these items, there would be greater clarity about where those items fit, how they would work, and what attributes and functions they would have. Looking at the object model system, the details of how the system will function are becoming clearer, but there is still more work to do before the code it written.