

Software Analysis and Design (CS:3004)

Entity Control and Boundary Classes

Use cases

- When someone wants a machine built, they tell us how they want the machine to behave when they use it. They give us examples, and we develop the machine's structure and control from the examples.
- An example behavior is called a *use case*.

Example: a use case of a candy-bar vending machine goes like this:

- (i) human inserts bill;
- (ii) human presses button with picture of candy bar;
- (iii) matching candy bar falls out of machine and onto floor.

A use case describes behavior from the user's perspective. But there is also behavior that occurs from the machine's perspective. When we include the latter into the use case, we have a *use-case realization* that gives us big clues how to build the machine that has the desired behavior.

For the candy machine, the use-case realization might go like this:

1.human inserts bill

The machine's bill slot rolls the bill under a scanner that checks the authenticity and the value of the bill.

The value is transmitted to the machine's money controller which enables the appropriate buttons for the candy bars.

The bill drops in the money box.

2.human presses button with picture of candy next to it

The button-press event notifies the button's controller for the selected candy bar.

The button's controller tells the data base to subtract the price of the bar from the amount deposited, and it also tells the candy-bar bin to release one bar (which it does --- we hope!)

A signal is sent to the change-box controller to issue coins for change (which it does --- we hope!)

Signals are sent to all button controllers to tell them to disable the buttons next to the pictures of the candy bars.

3.matching candy bar falls out of machine and onto floor

The realization talks about entity objects, both hard(ware) and soft(ware); multiple controllers for money and buttons and change; and buttons and pictures of the view (boundary entities).

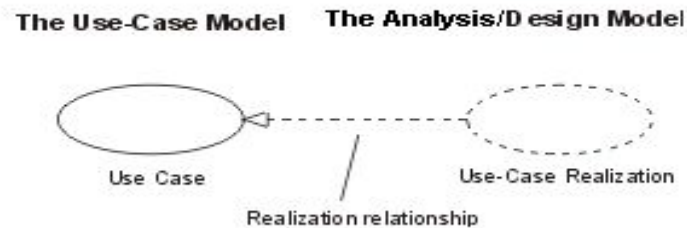
With enough use cases documented and realized, we can design a machine that does the behaviors.

When this technique is applied to a software system, it helps us design the class diagram that lets us build the system.

Use Case Realization

- A use-case realization represents how a use case will be implemented in terms of collaborating objects. The realizations reside within the design

The UML notation for use-case realization



The reason for separating the use-case realization from its use case is that doing so allows the requirements, in the form of use cases, to be managed separately from the design, in the form of realizations.

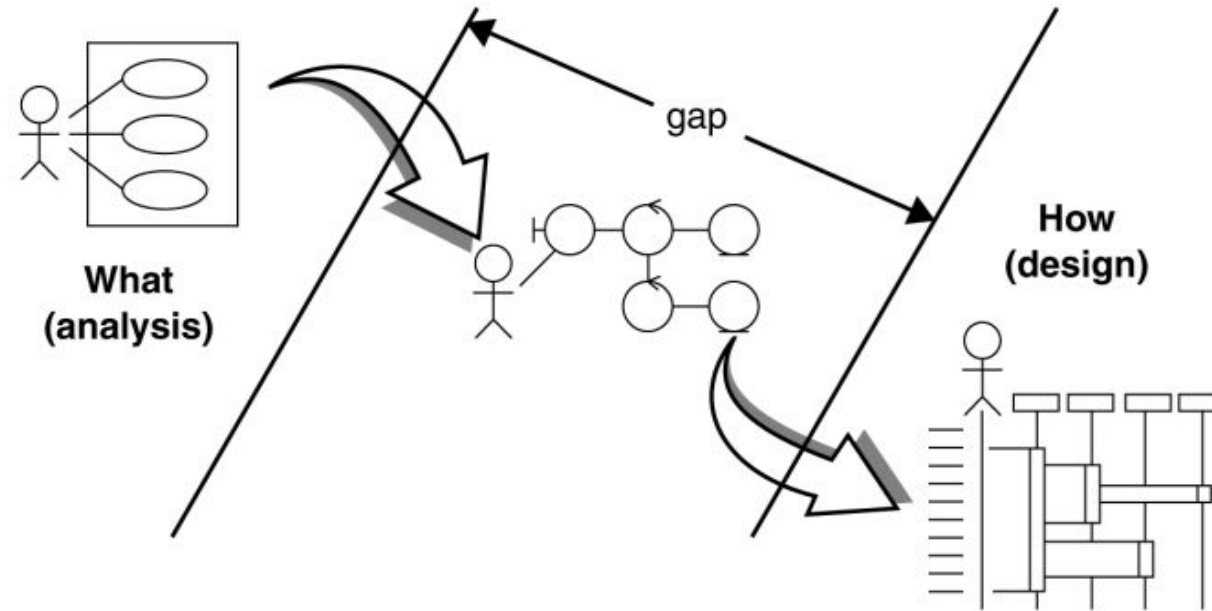
- According to RUP the Use Case Realization task creates the Analysis Model consisting of Boundary classes, Control classes and Entity classes diagrammed in Class Diagrams, and in interaction diagrams (Sequence and communication Diagrams).
- These Models are LOGICAL models describing white box inner workings of the system (as opposed to the use cases that treat the system as a Black box).
- The Design Model - that is often created by adding details and modifying the Analysis Model - represents the PHYSICAL description of the system to be done.

- One major difference between the Analysis Model and the Design Model, is that the Design Model contain implementation language specific information, whereas the Analysis Model does not.
- Use Cases are the WHAT, the functional requirements. Use Case Realizations are the HOW, the solution (at the logical level)
- Each use case realization will define the physical design in terms of classes and collaborating objects which support the use case. Therefore, each use case realization typically is made up of a class diagram and a number of interaction diagrams, most commonly sequence diagrams, showing the collaboration or interaction between physical objects.

ECB(Robustness Analysis):

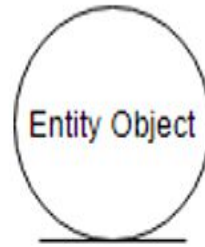
- The Robustness Analysis is a practice that originated with [Ivar Jacobson's](#) Objectory Method. (also called Jacobson's diagram)
- This involves analyzing the narrative text of use cases, identifying the first-guess set of objects that will participate in those use cases, and classifying these objects based on the roles they play.
- Robustness analysis helps you to bridge the gap from Use Cases and Domain Classes.

Robustness diagrams bridge the “what/how” gap



- ECB partitions the system into three types of classes: entities, controls, and boundaries.
- entity, control, and boundary are official UML class stereotypes. UML has some special icons to represent them.

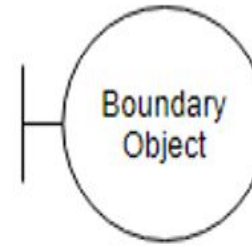
Robustness Analysis Diagram symbols:



Entity Object

Object
representing
stored data

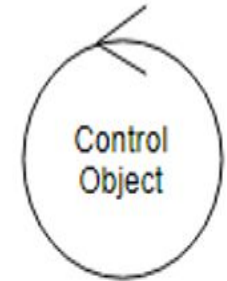
Model



Interface / Boundary Object

Object at the
system
Interface

View



Control Object

Object representing
transfer of
information

Controller

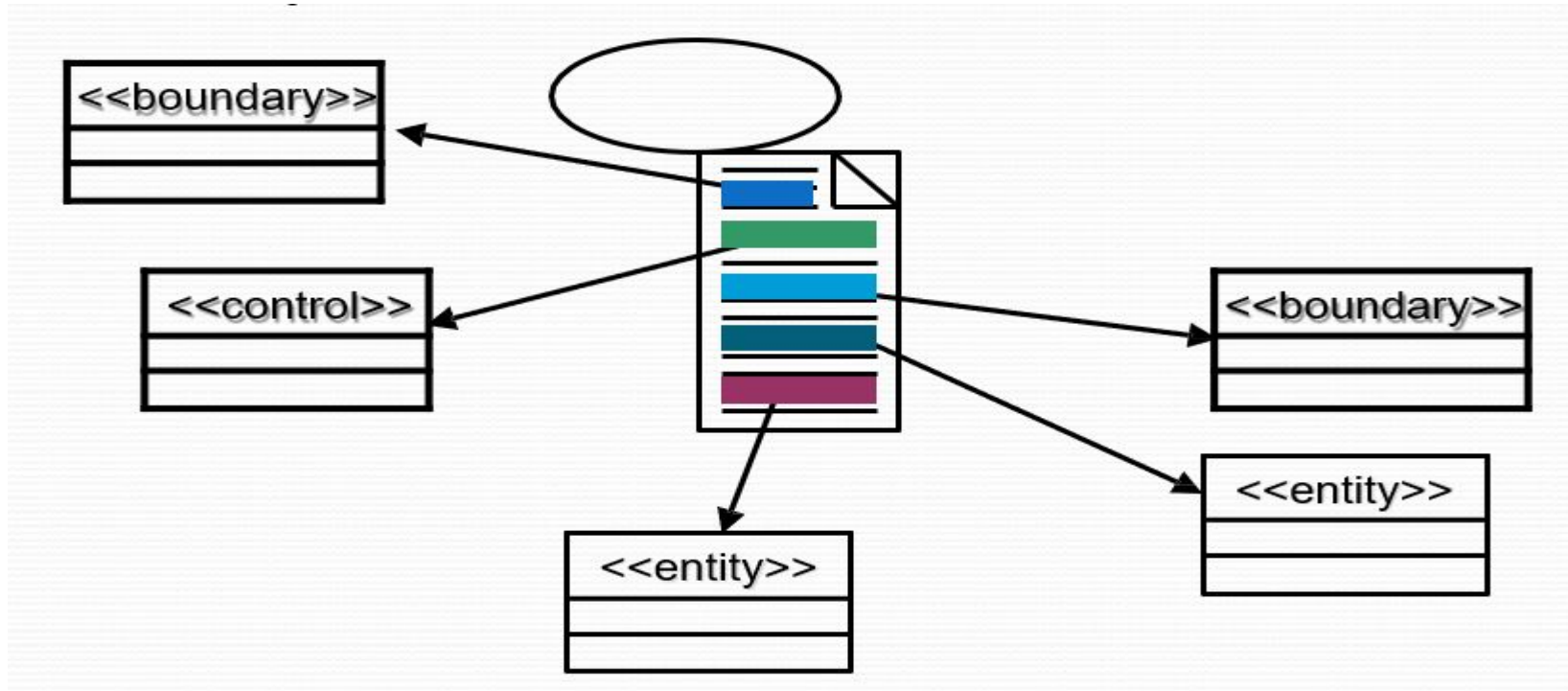
- **Boundary object** (or interface object) is what actors use in communicating with the system. The HTML, CSS, etc. which makes up the look and feel of the application
- **Entity object** The persistent data that we keep in the data store. Entities are objects representing system data: Customer, Product, Transaction, Cart, etc.
- **Control objects** (also known as controllers in MVC), which serve as the “glue” between boundary objects and entity objects. The code that does the thinking and decision making

What Is an Analysis Class?

- A class that represents initial data and behavior requirements, and whose software and hardware-oriented details have not been specified

Find Classes From Use-Case Behavior

- The complete behavior of a use case has to be distributed to analysis classes
- We must 'identify' these classes – identify, name, and briefly describe in a few sentences.



Discovering Classes

- Analysis classes represent an **early conceptual model** for ‘things in the system which have responsibilities and behaviors’.
- Analysis classes are used to capture a ‘first-draft’, rough- cut of the object model of the system.
- Analysis classes handle primary functional requirements, interface requirements, and some control - and model these objects from the problem domain perspective.

Kinds of Analysis Classes

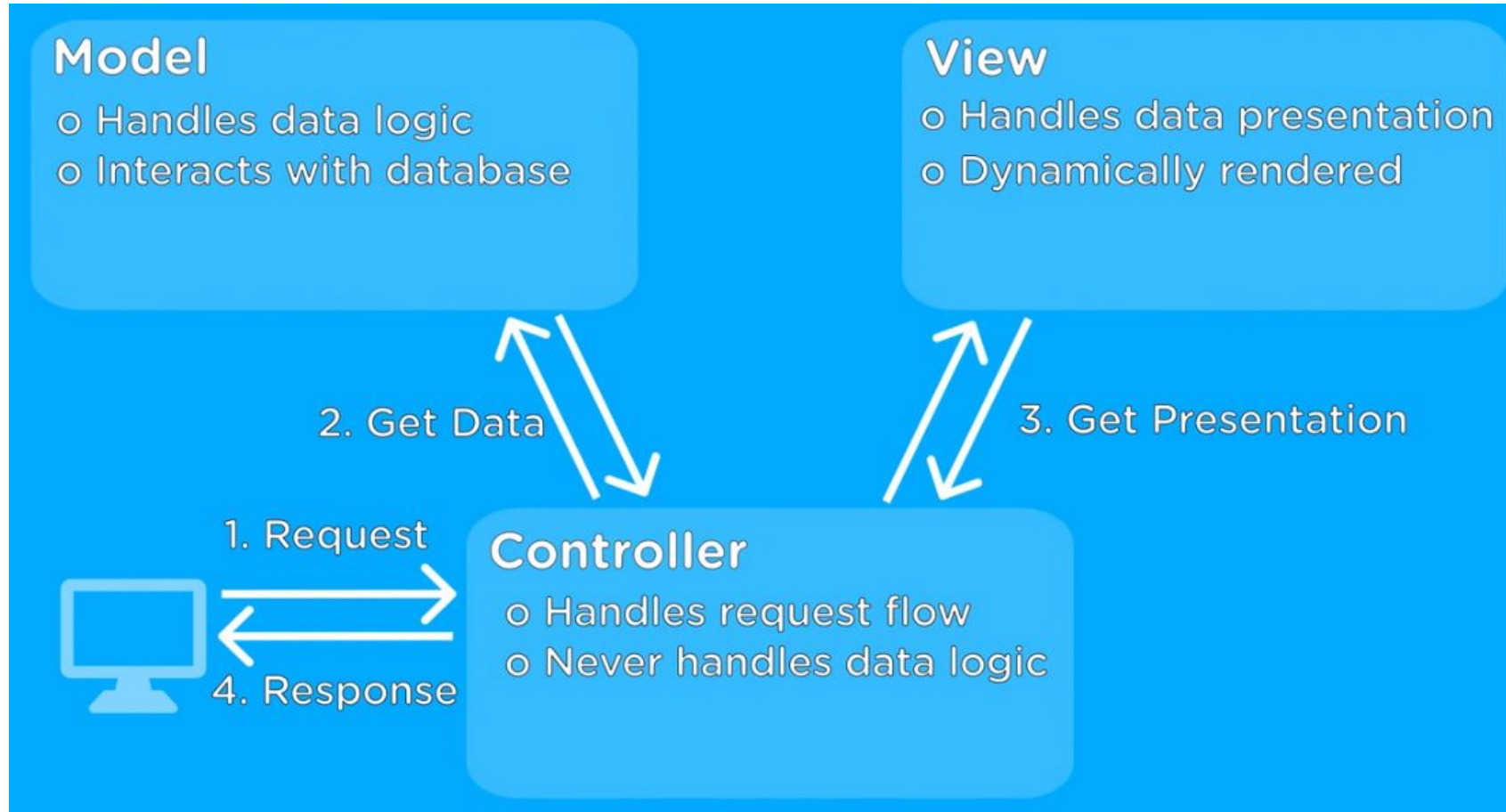
- Three things likely to change in a system:
 - The boundary between the system and its actors (interfaces...)
 - The information a system uses (data), and
 - The control logic of the system (who does what)
- So, we isolate the different kinds of analysis classes

Candidate Entity Classes

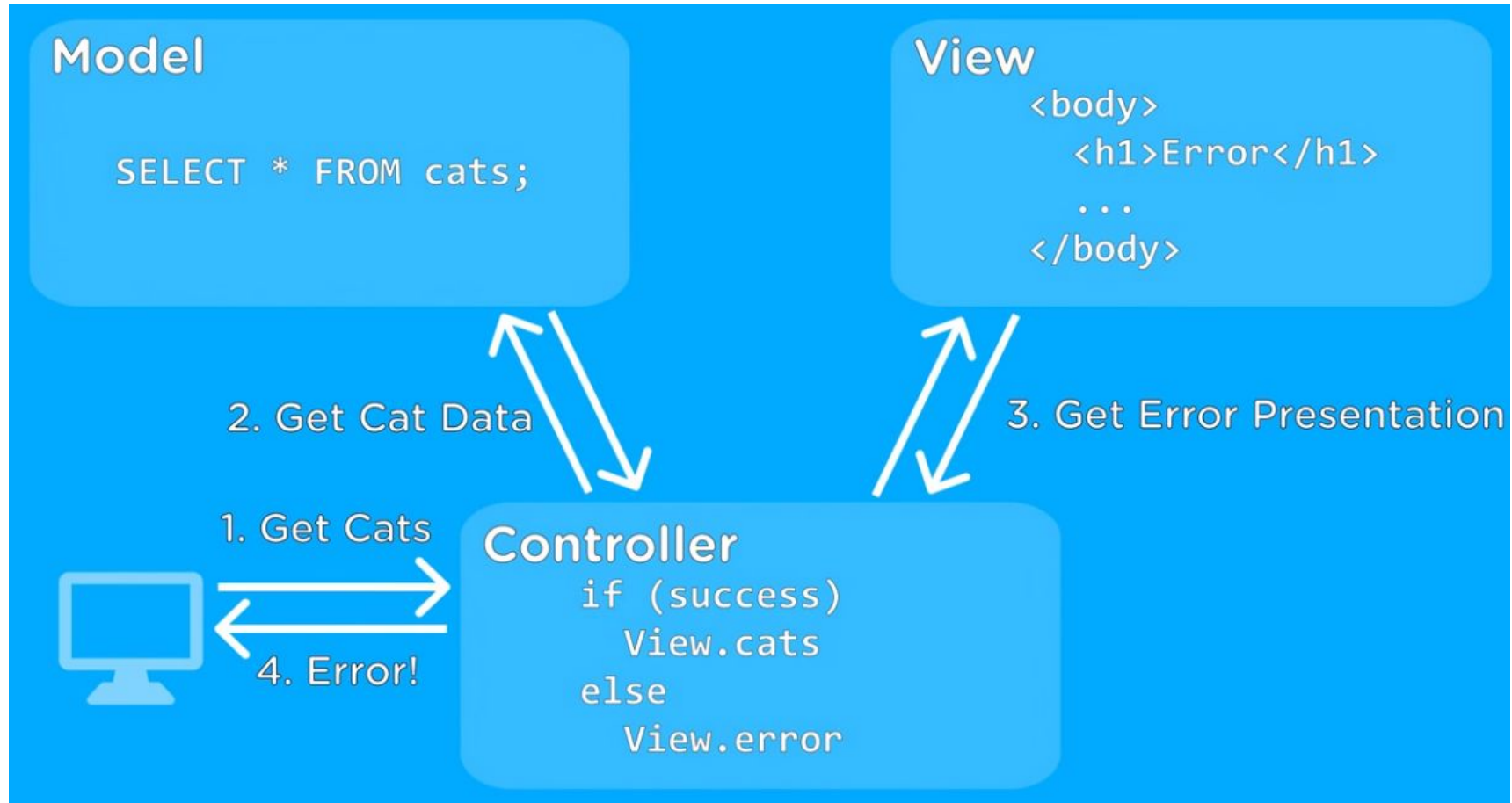
- Sometimes there is a need to model information about an actor within the system. This is not the same as modeling the actor (actors are external by definition). These classes are sometimes called “**surrogates**”.
- For example, a course registration system maintains information about the student which is independent of the fact that the student also plays a role as an actor of the system.
 - □ This information about the student that is stored in a ‘Student’ class is completely independent of the ‘actor’ role the student plays; the Student class (entity) will exist whether or not the student is an actor to the system.

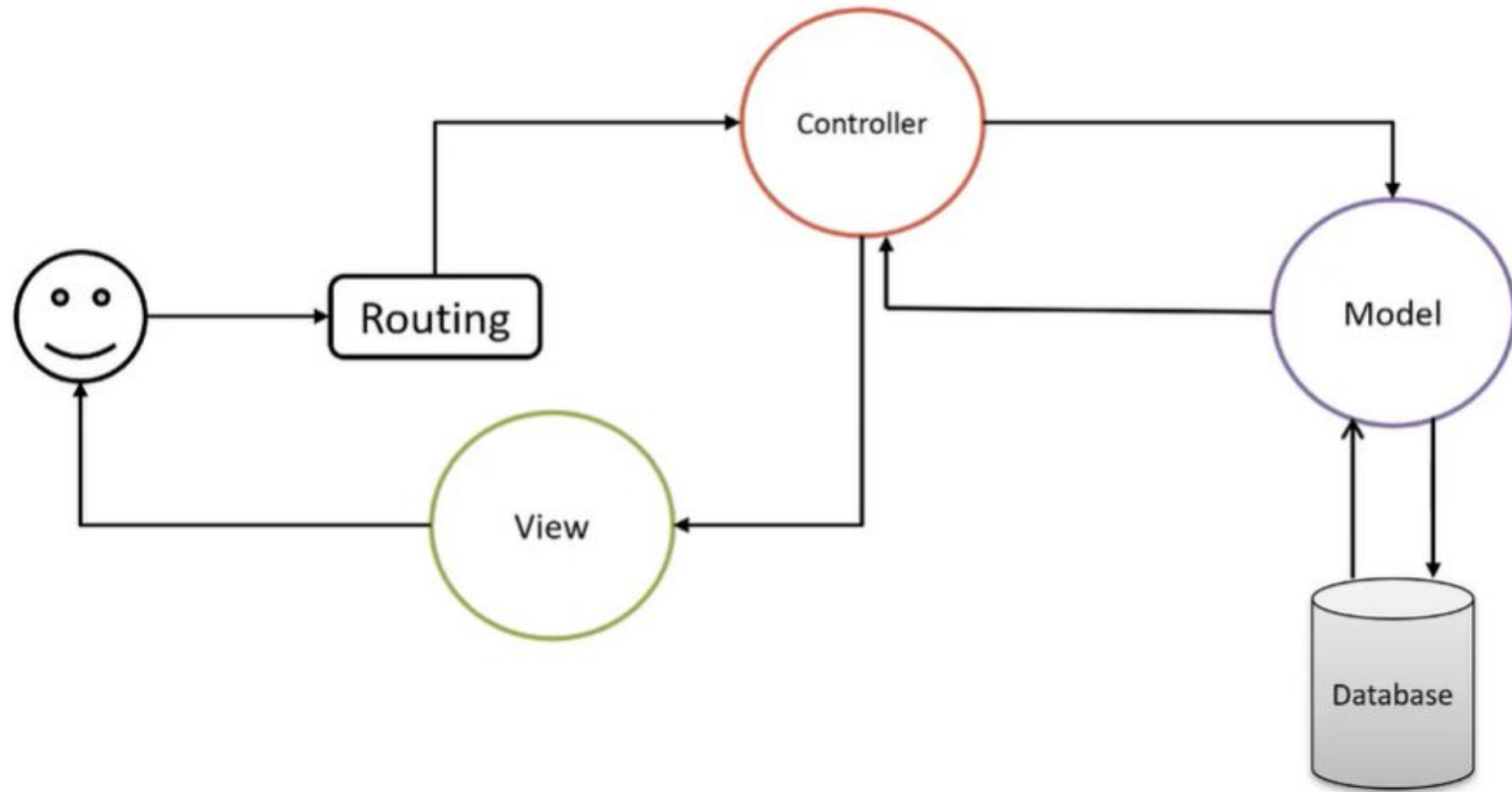
- Classes can be stereotyped (categorized) as
 - Control
 - Boundary
 - Entity.
- These stereotypes are used to increase the semantic meaning of the classes and their usage in modeling situations.
- These stereotypes are not found in the core of the UML specification. Rather, they are common stereotypes implemented in many UML tools and used during the design and analysis phase. They are based on the model-view-controller concept, where the entity is the model, the control is the controller, and the boundary is the view.

Model(entity) , Controller(control), View(boundary)



Model(entity) , Controller(control), View(boundary)



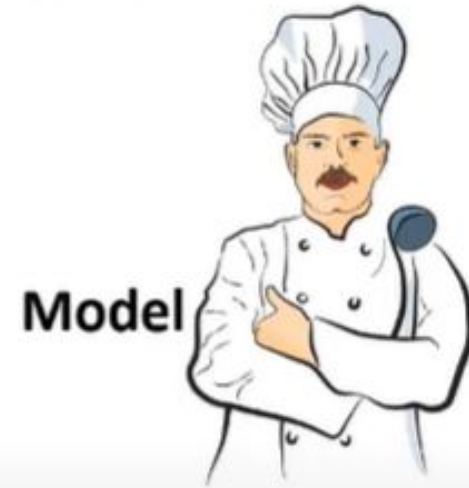




- Order Food
- Eat Food
- May order for food



- Take Order
- Pass Order info to Cook
- Receive Food from Cook
- Serve to Customer



- Access Vegetable
- Food



Several Types of Boundary Classes

User interface classes –

Classes that facilitate communication with human users of the system Menus, forms, etc. User interface classes.

System interface classes :

- Classes which facilitate communications with other systems.
- These boundary classes are responsible for managing the dialogue with the external system, like getting data from an existing database system or flat file.
- Provides an interface to that system for this system

Device Interface Classes

- Provide an interface to devices which detect external events – like a sensor or ...
- One boundary class per use case/actor pair

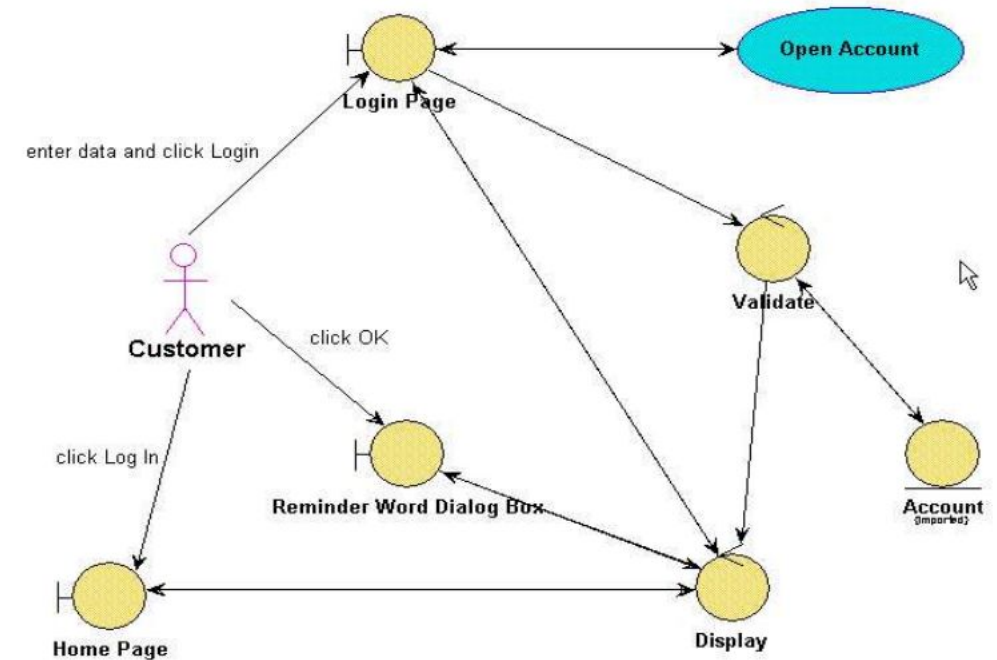
Robustness Analysis Diagram at a Glance:

Suppose we have the following simple use case description in textual format:

Login: use case text

Basic Course: The Customer enters his or her user ID and password, and then clicks the Log In button. The system validates the login information against the persistent Account data, and then returns the Customer to the Home Page.

Login: robustness diagram



Robustness Diagram – 4 Connection Rules

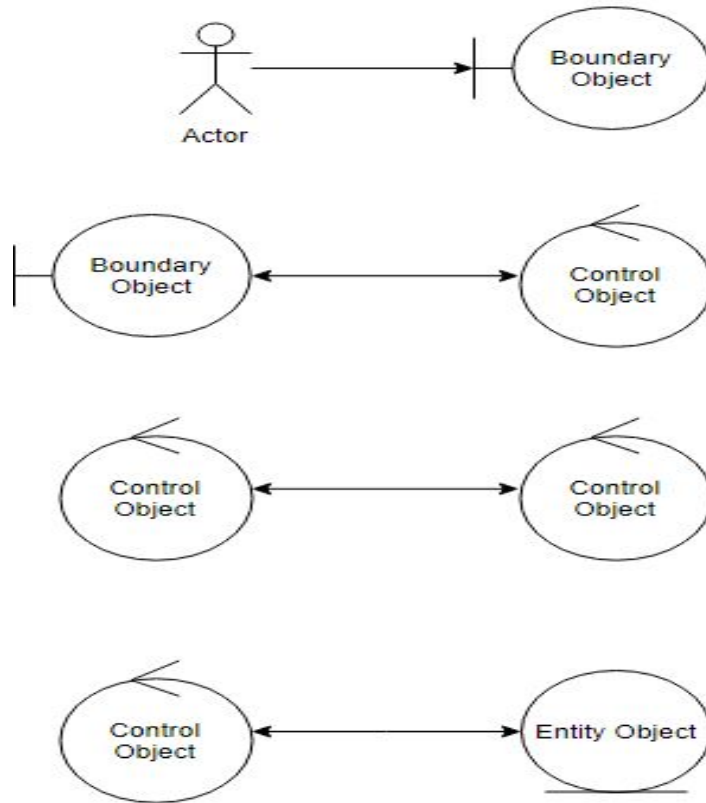
Keep in mind that both boundary objects and entity objects are nouns and that controllers are verbs. Nouns can't talk to other nouns, but verbs can talk to either nouns or verbs.

Here I listed the four basic connection rules which should always be mind:

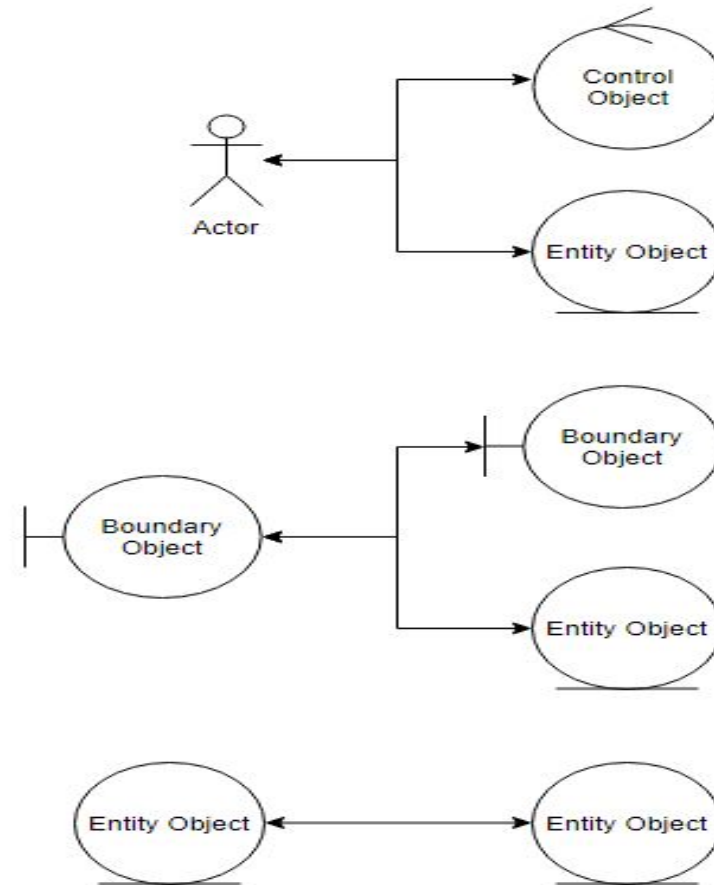
- Actors can only talk to boundary objects.
- Boundary objects can only talk to controllers and actors.
- Entity objects can only talk to controllers.
- Controllers can talk to boundary objects and entity objects, and to other controllers, but not to actors.

Robustness Analysis Diagram connection rules:





Allowed



Not Allowed

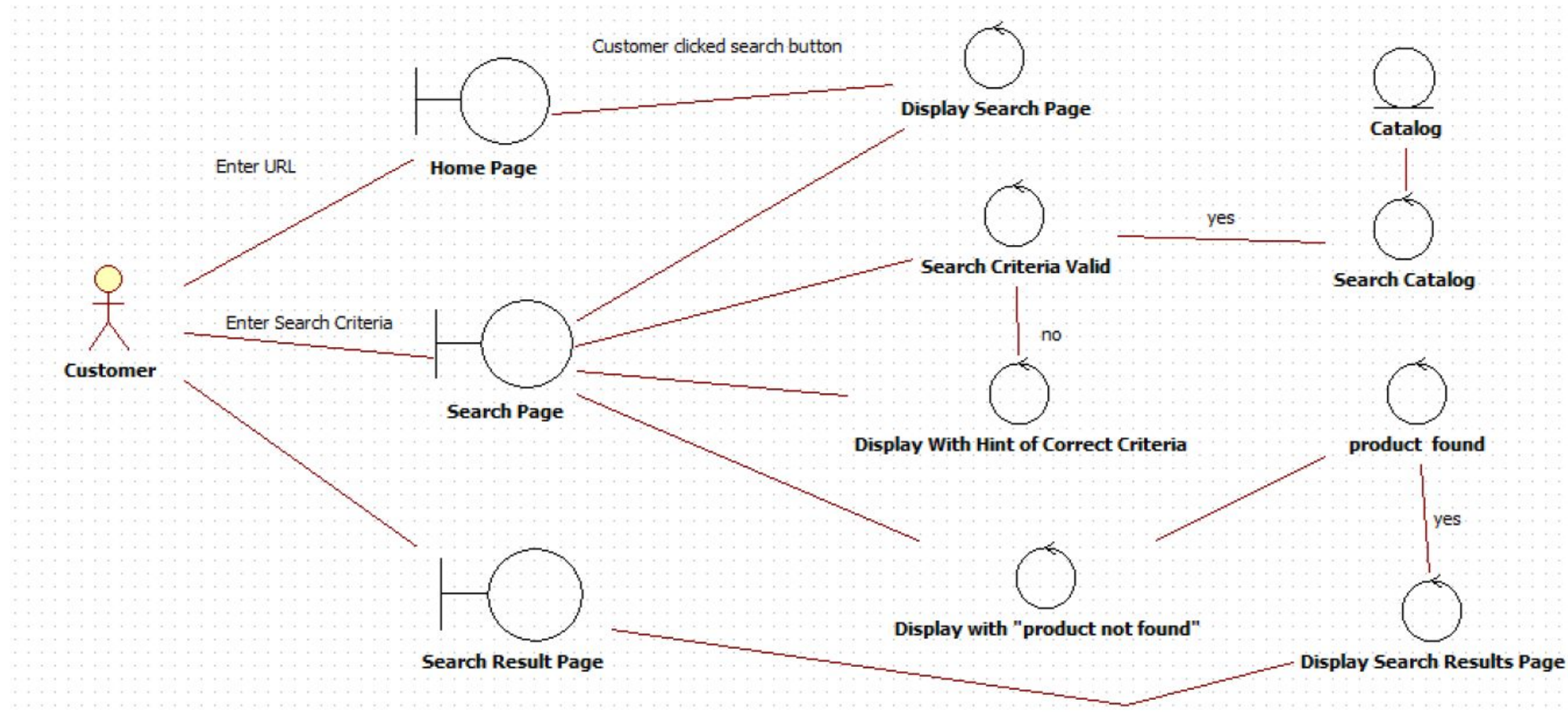


Robustness Analysis Diagram connection rules:

Representation		Relation with			
Role	Symbol	Actor	Boundary	Control	Entity
Actor		Yes	Yes	No	No
Boundary		Yes	Part/whole	Yes	No
Control		No	Yes	Yes	Yes
Entity		No	No	Yes	Yes

Use case ID	UC103
Use case Name	Search Products
Actors	Customer
Description	Search for products based on some criteria
	The customer wants to browse among products or the customer would like to search for certain products
Precondition	Customer starts web browser
Postcondition	Search results meeting the criteria are displayed
Normal Flow	<ol style="list-style-type: none">1. Customer visits the application home page2. Customer clicks the search button3. Search page is displayed by the system4. Customer enters search criteria5. The system validates the criteria provided6. The system looks up the catalog to find the products that meet the criteria7. Search results page is displayed with the product fulfilling the criteria
Alternative flows	<p>Refine Search Results</p> <p>The following steps are added:</p> <ol style="list-style-type: none">8. Customer refines search results by providing additional criteria9. Step 5-7 are re-executed
Exceptions	<p>In step 5, if search criteria is invalid or even missing then Step 3 (display search page) will be executed along with some hints on valid criteria</p> <p>Step 6, if no products found meet the criteria then Step 3 (display search page) will be executed along with the error message “Product not found”</p>
Included	None
Notes and issues	None

Robustness Diagram



Example 01.ECB example of a Course Registration System

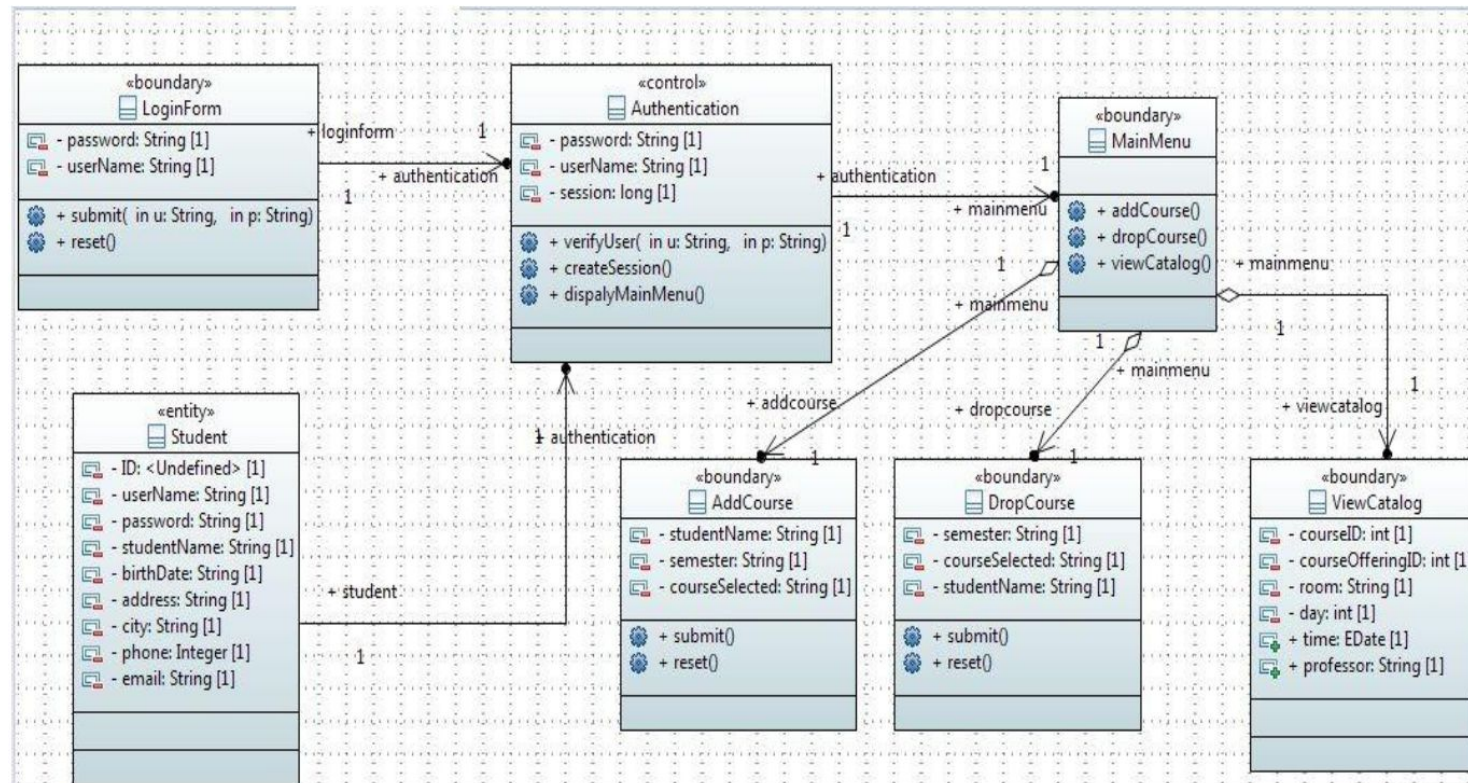


Figure 01

Login Sequence diagram:

Basic Course

The Customer clicks the Log In button on the Home Page. The system displays the Login Page. The Customer enters his or her user ID and password, and then clicks the Log In button. The system validates the login information against the persistent Account data, and then returns the Customer to the Home Page.

Alternate Courses

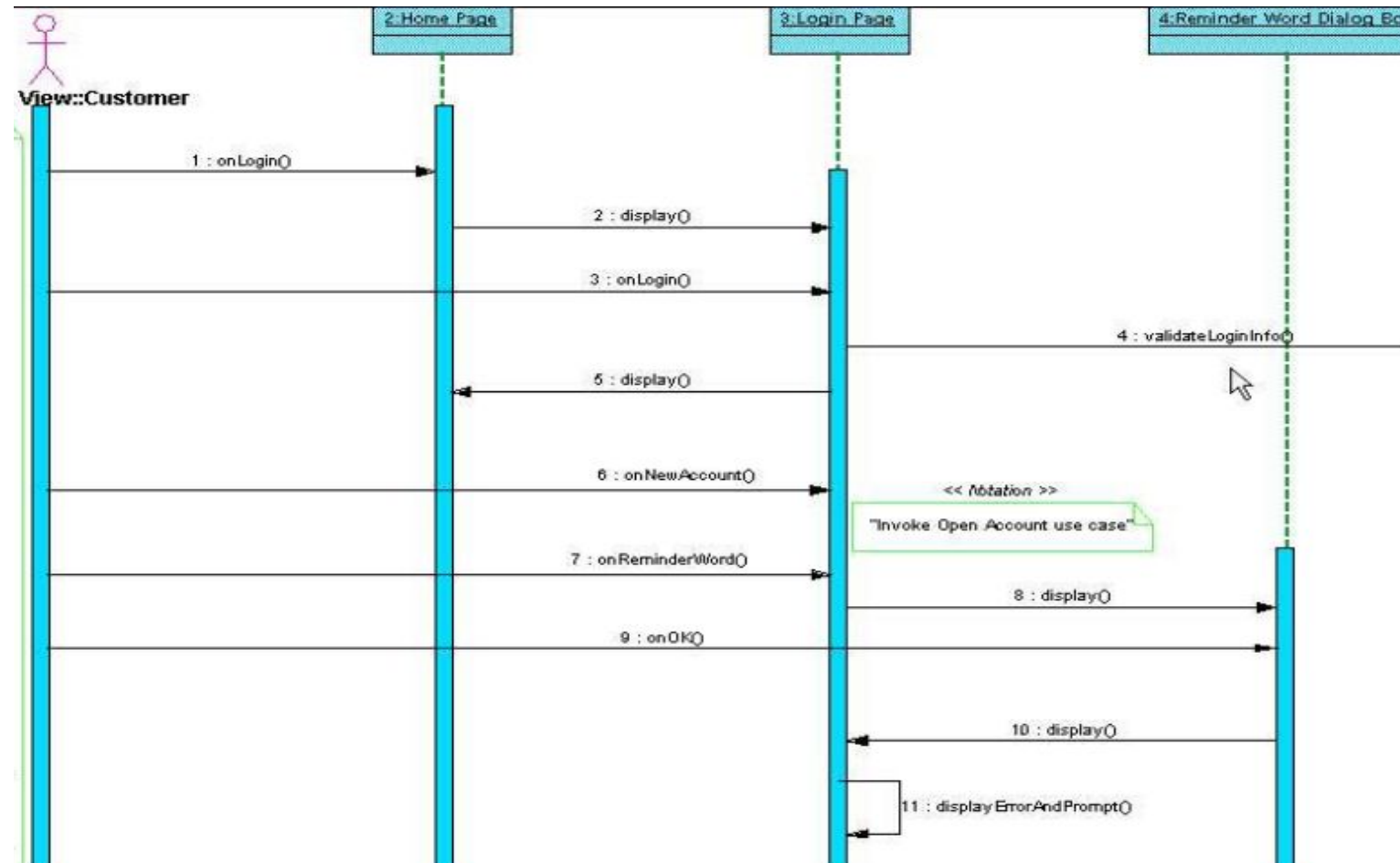
If the Customer clicks the New Account button on the Login Page, the system invokes the Open Account use case.

If the Customer clicks the Reminder Word button on the Login Page, the system displays the reminder word stored for that Customer, in a separate dialog box. When the Customer clicks the OK button, the system returns the Customer to the Login Page.

If the Customer enters a user ID that the system does not recognize, the system displays a message to that effect and prompts the Customer to either enter a different ID or click the New Account button.

If the Customer enters an incorrect password, the system displays a message to that effect and prompts the Customer to reenter his or her password.

If the Customer enters an incorrect password three times, the system displays a message



Edit Shopping Cart use case text:

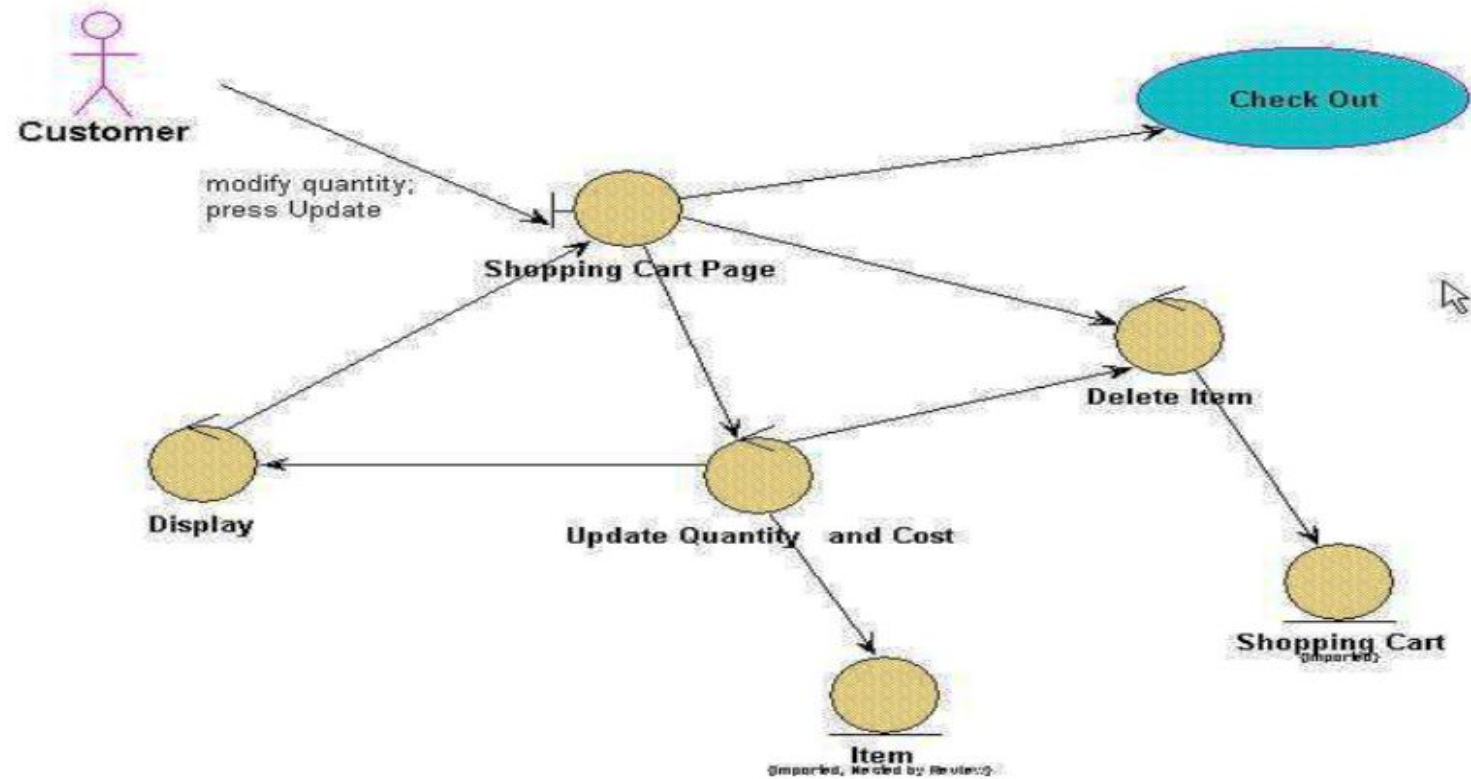
Basic Course:

- On the Shopping Cart Page, the Customer modifies
- the quantity of an Item in the Shopping Cart, and then presses the
- Update button. The system stores the new quantity, and then
- computes and displays the new cost for that Item.

Alternate Course:

- If the Customer changes the quantity of the Item
- to 0, the system deletes that Item from the Shopping Cart.

Edit shopping cart robustness diagram



Five Steps for Creating Robustness Analysis:

- You perform robustness analysis for a use case by walking through the use case text.
- One sentence at a time, and drawing the actors, the appropriate boundary, entity objects and controllers, and the connections among the various elements of the diagram.
- You should be able to fit the basic course and all of the alternate courses on one diagram.
- Anyone who reviews a robustness diagram should be able to read a course of action in the use case text, trace his finger along with the associations on the diagram, and see a clear match between text and picture.

Checkout

Basic Course:

The system displays the Edit Shopping Cart page. The user clicks the Checkout button; the system displays the Delivery Address page showing any addresses already registered for the user's account. The user selects the address; the system sets the delivery address in the order.

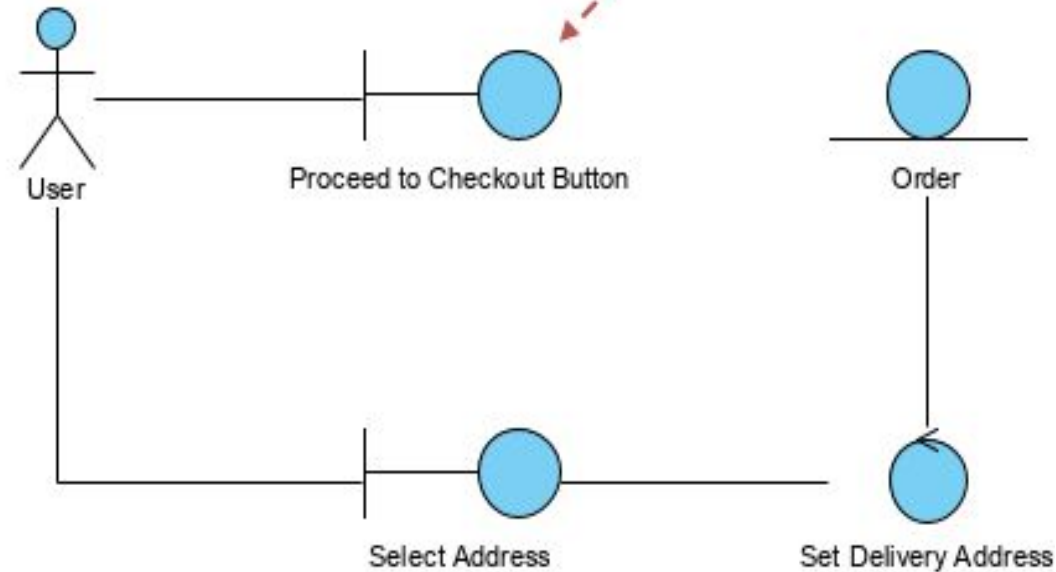
...

Ambiguous use case text

The start of the use case text doesn't match up with the diagram

This interaction is missing from the diagram

UI element shown as boundary object



The real boundary object is missing