MET CS 682 INFORMATION SYSTEMS ANALYSIS AND DESIGN: ASSIGNMENT 4


# Discussion Boards – Models

**Nicholas Scala**
**February 2016**

## Objective

This project artifact serves as a partial, preliminary system design document for *Blackboard Learn* Discussion Boards. It contains a use case model, sequence diagram, class model, and activity diagram modeling when a user posts to an existing thread.

# TABLE OF CONTENTS

# I. Use Case

Below is a use case for posting to an existing thread in a *Blackboard Learn* discussion board:

| **Use Case 002: User Posts to an Existing Thread**[1] | | |
|---|---|---|
| **ID** | 002[2] | |
| **Brief description** | As a Blackboard User, I want to post to an existing thread in *Blackboard Learn*, so that I can share my thoughts on a topic. | |
| **Primary actors** | Blackboard User | |
| **Secondary actors** | *(none)* | |
| **Pre-conditions** | The user must be logged in.<br>The course must have discussions enabled.<br>There must be a pre-existing thread of interest to the user that they wish to post to. | |
| **Trigger** | This use case is initiated when the user enters the "Discussion Board" section of *Blackboard Learn* with the intention of posting to an existing thread. | |
| **Main flow** | **Actor** | **System** |
| | 1. User selects desired "Forum" | 2. System displays list of "Threads" |
| | 3. User selects desired "Thread" | 4. System displays list of "Posts" |
| | 5. User selects "Reply" on desired post | 6. System displays "Post Editor" box |
| | 7. User selects the "Subject" field and types in a new subject | 8. System displays user's input as the updated subject |
| | 9. User selects in the "Message" field and types in a new message | 10. System displays user's input as the updated message |
| | 11. User selects "Browse My Computer" to attach a file | 12. System displays local "File Browser" |
| | 13. User selects desired file | 14. System commits file to the server and displays title of uploaded file |
| | 15. User selects "Submit" | 16. System commits post to the server, displays confirmation, and displays post |
| **Post-conditions** | Post is saved, displayed successfully, and user receives confirmation of such | |
| **Alternate flows** | The user can cancel their request or exit the webpage at any time before submission. Additional alternative flows and details, such as emailing the author or saving a draft are displayed in the Activity Diagram below. | |

[1] Use case template derived from Betty Leudke with additions from Whitten and Bentley (refer to references for detailed citation).

[2] It is assumed that this use case would be one of many for this system, thus it is given a dummy ID.

# II. Entity Classes

Below is a list of entity classes essential to the implementation of this software. These entity classes hold system data necessary for carrying out the use case described above. (Pearce, n.d.). The list of classes below was originally identified by first understanding, "what are the most important things that will be worked within the system?" (Shvets, n.d.) After the sequence diagram was constructed, this list was updated to keep both diagrams consistent as additional classes were discovered.

## Entity Classes

| Class Title | Purpose | Attributes | Functionalities | Relationships[3] |
|---|---|---|---|---|
| **User** | This class is used to store all information regarding the user so that each discussion board action may be identified by the user. | • userID<br>• userFirstName<br>• userLastName<br>• userEmail<br>• userType | The User class will be able to view, create, edit, and delete user information. | One **user** can create multiple **posts**.<br>Each **post** is created by one **user**. |
| **Instructor, Facilitator, Student**[4] | These classes are specializations of the User class to account for unique functions each role may have. | Given the inheritance relationship, these classes will inherit all the attributes from the User class and have additional attributes if necessary. | Given the inheritance relationship, these classes will inherit all the functionalities from the User class and have additional functionalities if necessary. | **Instructure**, **Facilitator**, and **Student** are specializations of the **User** class. The User class is a generalization of these classes. |
| **Forum** | This class is used to store all information regarding the forum. | • forumID<br>• forumTitle<br>• threadList | The Forum class will be able to view, create, edit, and delete forum information, and display the list of threads. | One **forum** can have many **threads**. Each **thread** belongs to one **forum**. |
| **Thread** | This class is used to store all information regarding the thread. | • threadID<br>• threadTitle<br>• postList | The Thread class will be able to view, create, edit, and delete thread information, and display the list of posts. | One **thread** can have many **posts**. Each **post** belongs to one **thread**. |
| **Post** | This class is used to store all information regarding the post. This includes not only post identification, but also whether or not a post is an original or a reply, | • postID<br>• postTitle<br>• postReply[5]<br>• postTimeStamp<br>• postStatus[6] | The Post class will be able to view, create, edit, and delete post information. | One **post** is created by one **user**. One **user** can create many **posts**.<br>One **post** can have one **content** block. Each **content** block belongs to one **post**. |

[3] Simple relationships with multiplicities are described here for conciseness. For aggregation, composition, specialization, and additional relationships, please refer to the Class Model in Section IV.

[4] These classes have been added and grouped together to ensure consistency with the additional instructions provided in Class Model activity.

[5] This field is used to determine if the post is an original post or a reply as one original post can have many replies.

[6] This field is used to determine if the post is in a draft or final state.

| | | | | |
|---|---|---|---|---|
| | and whether or not the post is final or a draft. | | | One **post** can have one **file**. Each **file** can belong to many **posts**[7]. |
| | | | | One **post** can have many **replies**. Each **reply** can belong to one **post**. |
| **Content** | This class is used to store all information regarding the content. | • contentID<br>• contentText | The Content class will be able to view, create, edit, and delete content information. | One **content** block belongs to one **post**. Each **post** can have one **content** block. |
| **File** | This class is used to store all information regarding the post. | • fileID<br>• fileTitle<br>• fileLocation | The File class will be able to view, create, edit, and delete file information. | One **file** can belong to many **posts**. Each **post** can have one **file**. |

# Non-Entity Classes

There are four non-entity classes that would make it into the final design. These classes are not entity classes because they are not, "something about which the business needs to store data" (Whitten and Bentley, 2007, pg. 271), but are still essential to the system.

| Class Title | Non-Entity Class Type | Purpose |
|---|---|---|
| **BlackBoard UI** | <<interface>><br>*(see note 1)* | From logging in and logging out, to displaying the university's logo and discussion board list, this interface class drives the vast majority of the user's interactions throughout the system. |
| **PostEditorUI** | <<interface>> | This interface class serves as the editor in which users can input and format text, insert pictures and videos, and add hyperlinks amongst other content-related activities. |
| **FileUploadUI** | <<interface>> | This interface class serves as window in which users can browse for and attach files to a post. |
| **CreateNewPost** | <<controller>><br>*(see note 2)* | This controller class serves as the intermediary between the interface classes and the entity classes for the "User Posts to an Existing Thread" use case. |

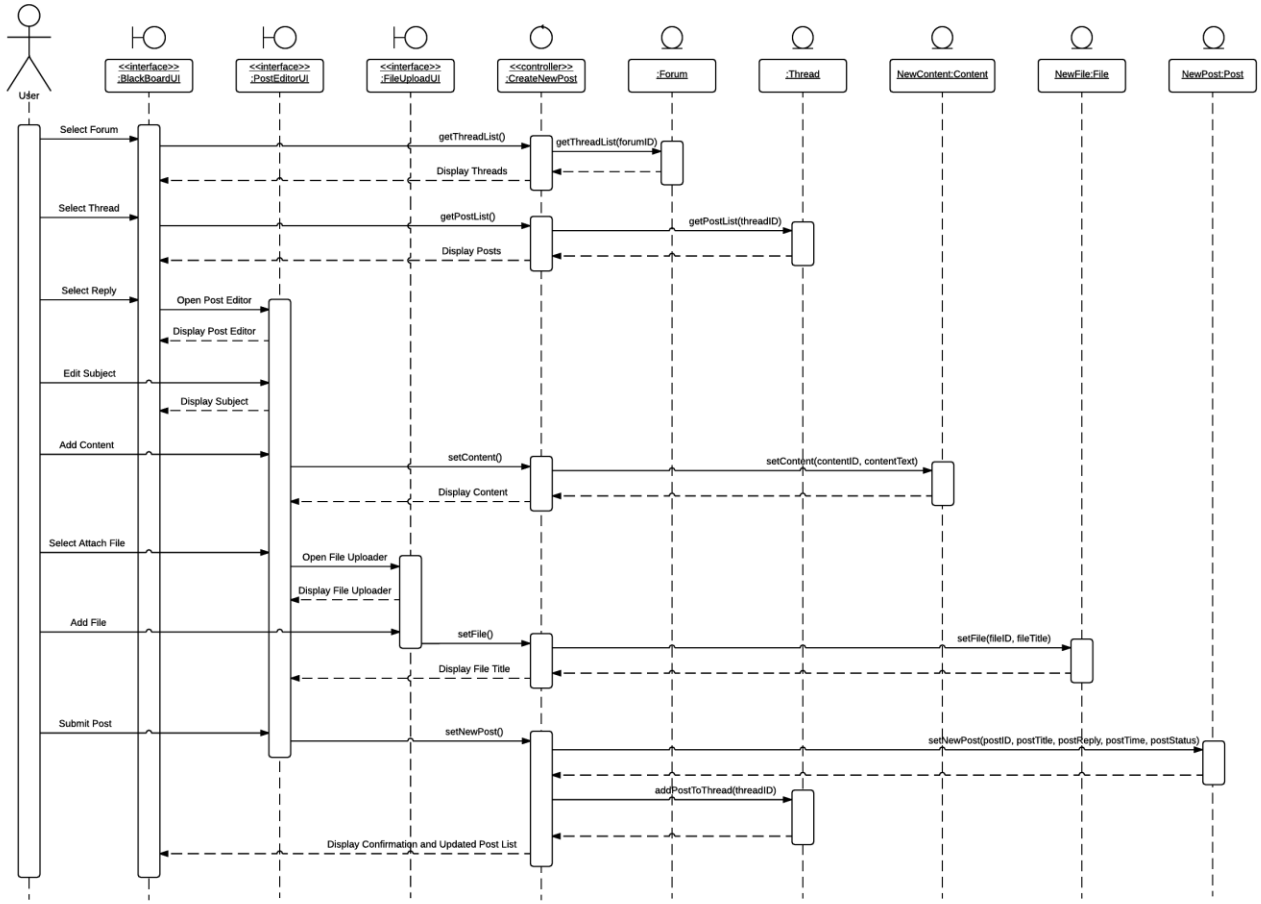
**Note 1**: Interface classes serve two purposes – (1) they, "translate the user's input into information that the system can understand and use to process the business event" and (2) they, "take data pertaining to a business event and translate the data for appropriate presentation to the user." (Whitten and Bentley, 2007, pg. 649) These classes drive the user's interaction with the system including displaying buttons, forms, colors, etc.

**Note 2:** Control classes, "process messages from an interface class and respond to them by sending and receiving messages from me entity classes." (Whitten and Bentley, 2007, pg. 649). These classes, sometimes referred to as "boundary" classes, serve as the middleman between the interface and entity classes. They hold all the business logic, but do not contain the necessary data and thus interact with the entity classes to obtain the data.

[7] Using the "Content Collection" feature in *Blackboard Learn* it is possible that one file can belong to many posts. If the user is uploading a file from their local drive using the "Browse My Computer" feature, then one file is associated with only one post.

# III. Sequence Diagram

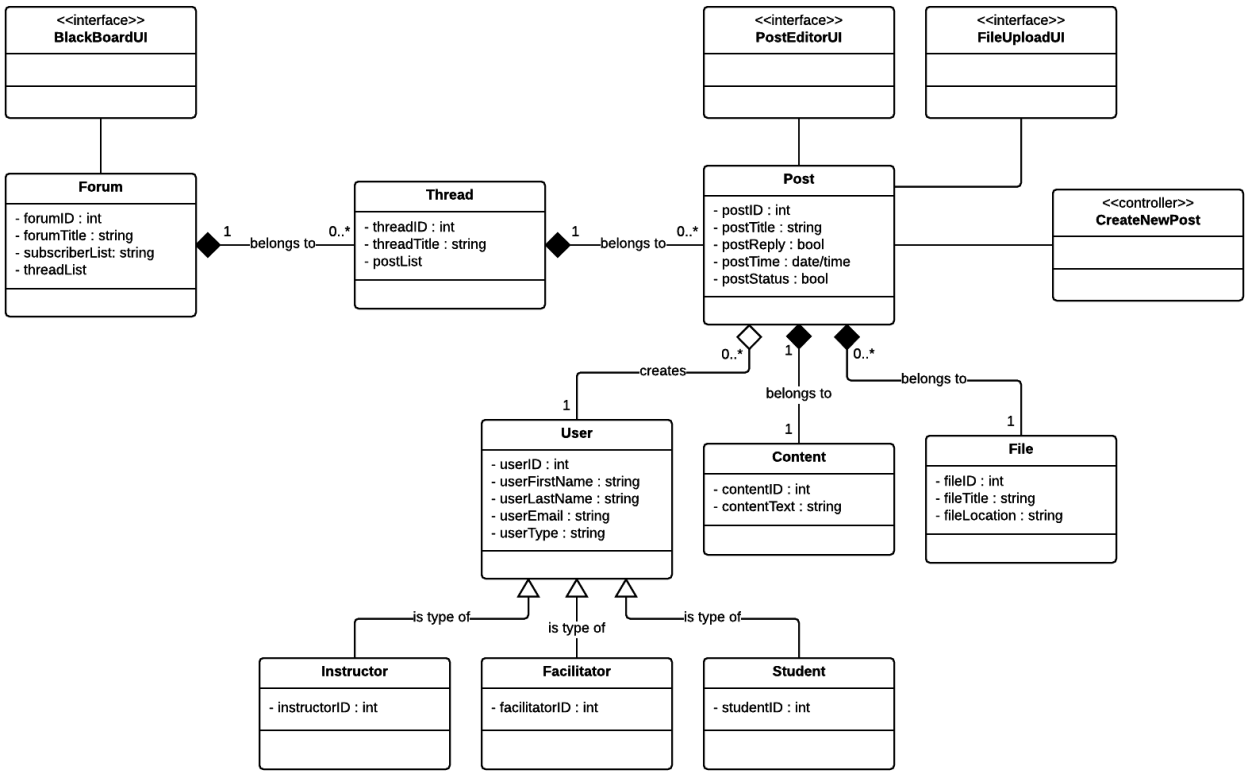
Below is a sequence diagram for posting to an existing thread in a *Blackboard Learn* discussion board:




The decisions for the sequence diagram above were heavily influenced by Braude's Module 4 lecture notes and the example on page 659 of the Whitten and Bentley textbook (refer to references for full citation). The user actions for the diagram were derived to match the main flow of the use case. The elements for the diagram are based on the classes, attributes, and behaviors outlined in the entity classes description and the class model. The sequence diagram also adheres to the "Model/View/ Controller" principles (Alexander, 2014) where the user interacts with interface classes, the interface classes interact with controller classes, and the controller classes interact with the entity classes as denoted by the symbols above the classes.

# IV. Class Model

Below is a class model for the *Blackboard Learn* discussion board:



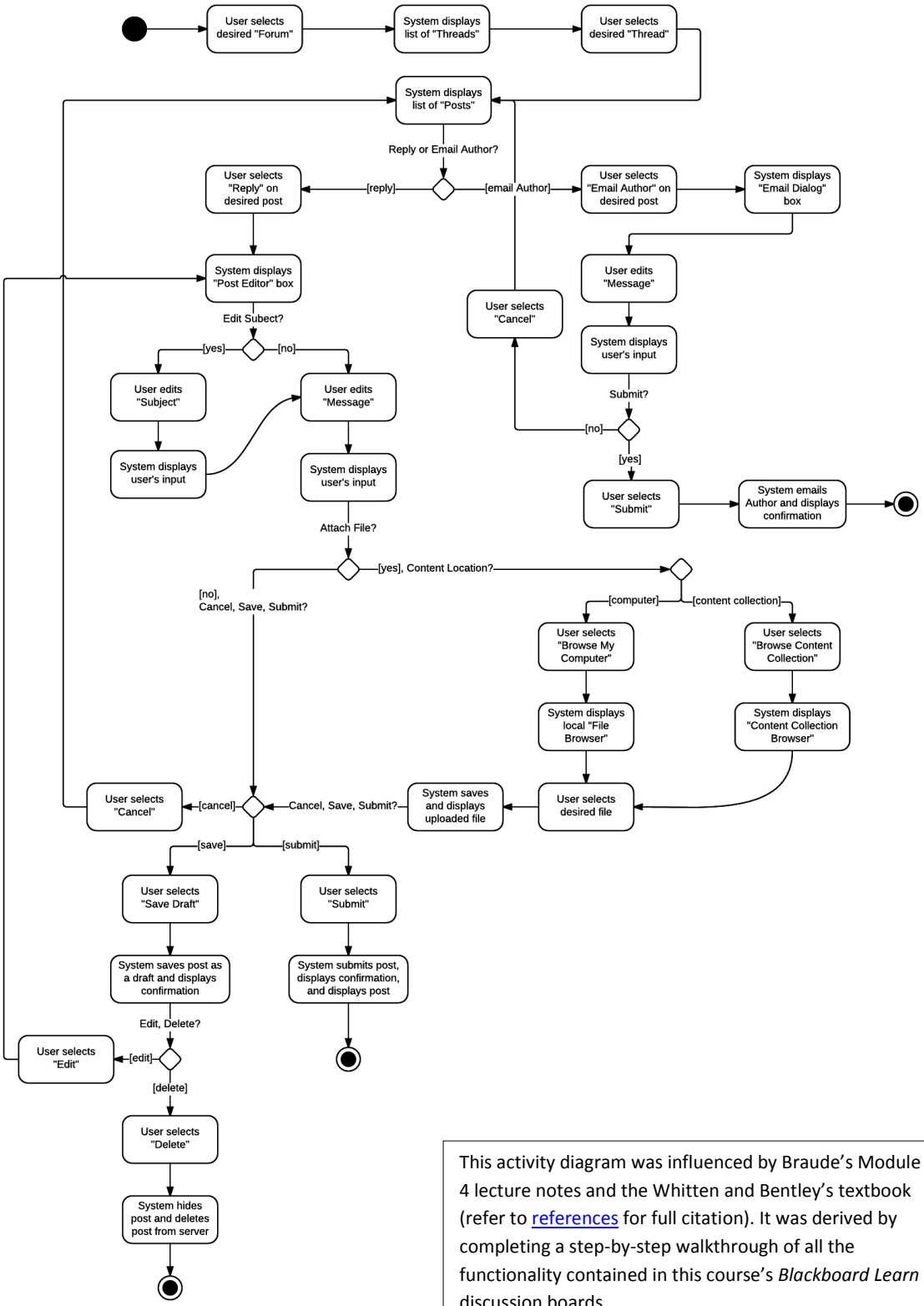

The above class model was influenced by Braude's Module 4 lecture notes and the Whitten and Bentley's textbook (refer to references for full citation). Methods were excluded from this class model for simplicity. This class model was updated to included interface and controller classes after constructing the sequence diagram to ensure consistency of the end-to-end solution.

Relationships:

- There exists an **aggregation** relationship between User and Post because if a post is to be deleted, the user information can and should still remain in the system.
- There are **composition** relationships between Forum and Thread, Thread and Post, Post and Content, and Post and File because, for example, when a forum is removed, so to are the threads belonging to that forum, the posts belonging to those threads, and the content and files associated with those posts. (Alexander, 2014)
- There are **inheritance** relationships between User and Instructor, Facilitator, and Student because the child classes (Instructor, Facilitator, and Student) will, "have common attributes and/or operations" with the parent class (User), but will also have differing characteristics as described by the instructions. (Braude, 2016, Section 3.3)

# V. Activity Diagram

Below is an activity diagram for posting to an existing thread in a *Blackboard Learn* discussion board:




This activity diagram was influenced by Braude's Module 4 lecture notes and the Whitten and Bentley's textbook (refer to references for full citation). It was derived by completing a step-by-step walkthrough of all the functionality contained in this course's *Blackboard Learn* discussion boards.

# References

Alexander, A. (2014, March 23). A terrific Model View Controller (MVC) diagram. Retrieved February 14, 2016, from http://alvinalexander.com/uml/uml-model-view-controller-mvc-diagram.

Braude, Eric J. (2016). Module 4 Class Notes [PDF]. Retrieved February 12, 2016, from https://onlinecampus.bu.edu/webapps/blackboard/execute/displayLearningUnit?course_id=_29083_1&content_id=_3742163_1&framesetWrapped=true.

Ezra, A. (2009, May 28). UML Class Diagram: Association, Aggregation and Composition. Retrieved February 14, 2016, from http://aviadezra.blogspot.com/2009/05/uml-association-aggregation-composition.html.

Pearce, J. (n.d.). The Entity-Control-Boundary Pattern. Retrieved February 14, 2016, from http://www.cs.sjsu.edu/~pearce/modules/lectures/ooa/analysis/ecb.htm.

Shvets, A. (n.d.). Design Patterns and Refactoring. Retrieved February 14, 2016, from https://sourcemaking.com/uml/modeling-it-systems/structural-view/constructing-class-diagrams.

Whitten, J., & Bentley, L. (2007). Systems analysis and design methods (7th ed.). Boston, Mass.: Irwin/McGraw-Hill.

---

Cover image retrieved from: https://sites.newpaltz.edu/tlcdev/wp-content/uploads/sites/46/2013/03/Blackboard_Learn.jpg.

Diagrams created using Lucid Chart accessible here: https://www.lucidchart.com/.