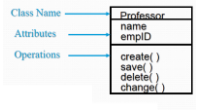
ECB Pattern & Robust Analysis

Topic # 10

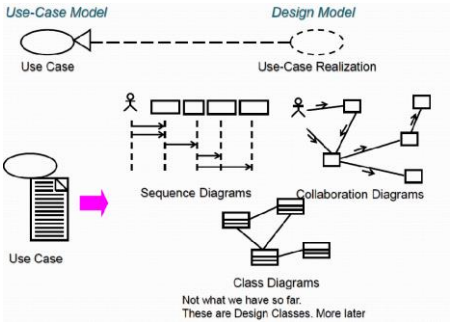
Review: Class

- Recall: A class is an abstraction
- It describes a group of objects with common:
  - Properties (attributes)
  - Behavior (operations)
  - Relationships
  - Semantics



- A class is an abstraction in that it:
  - Emphasizes relevant characteristics, suppresses others
  - Consists of three sections:
    - First section: Class name
    - Second section: structure (attributes)
    - Third section: behavior (operations)
- For analysis classes, these entries are sufficient

Use Case Realization



Use Case Realization

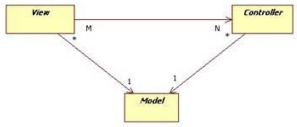
- Within the design model, a use case realization describes how a particular use case is implemented, in terms of collaborating objects.
  - A use case realization is **one possible realization** of a use case.
  - A use case realization in the design model can be traced to a use case in the use case model
  - A realization relationship is drawn from the use case realization to the use case in the use case model
- Use case realization can be represented using set of diagrams (the number and type may vary).

Use case Realization

- The diagram that may be used to realize a use case realization may include:
- **Class Diagrams** can be used to describe the classes that participate in the realization of the use case, as well as their supporting relationships.
  - These diagrams model the context of the use case realization.
- **Interaction Diagrams** (sequence and/or collaboration diagrams) can be used to describe how the use case is realized in terms of collaborating objects.
  - These diagrams model the **detailed collaborations** of the use case realization.

ECB Pattern(Implementing use cases)

- The Entity-Control-Boundary Pattern (ECB) is a variation of the Model-View-Controller Pattern.
- MODEL VIEW CONTROLLER:
  - Application data and logic (encapsulated by the model) should be independent of presentation logic (encapsulated by the views and controllers):
  - Views are responsible for user input and output.
  - The model encapsulates the fine grained business logic and data.



Use Case Realization

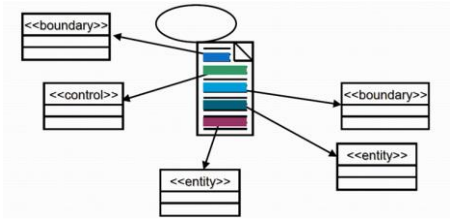
- In **Use Case Analysis** step, the use-case realizations' diagrams are outlined.
- In subsequent **design activities** (Use Case Design), these class diagrams will be considerably refined and updated according to more formal class interface definitions.

Identifying candidate classes from behavior

- Will use three perspectives of the system to identify these classes.
  - The 'boundary' between the system and its actors.
  - The 'information the system uses.
  - The 'control logic' of the system.
- Will use stereotypes (boundary, control, entity) to represent these perspectives.
  - These are conveniences used during analysis that will disappear or be transitioned into different design elements during the design process
- Will result in a more robust model because these are the three things that are most likely to change in the system and hence we isolate them so that we can treat them separately.
  - That is, the interface/boundary, the control and the key system entities.

Investigating 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.

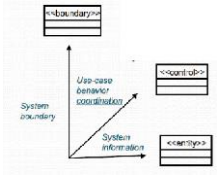


Classes Discovery

- 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 primarily the functional requirements, interface requirements and some control - Analysis classes model objects from the problem domain perspective.

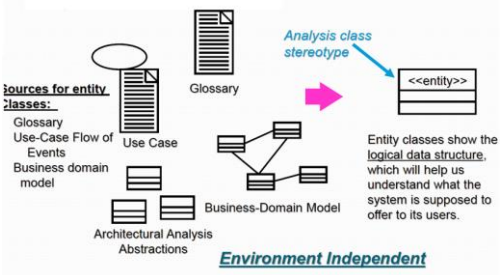
Class Diagrams – From Analysis to Design

- Design will talk about implementations and UI details and elaborate some more logics of control logic.
- Finding a candidate set of classes is the first part of transforming a mere statement of required behavior to a description of how the system will work.
  - 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)
- Can use with the name of the stereotype or as symbols with unique icons.



Entity class

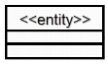
- Key abstractions of the system:

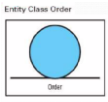


Entity Class

- **Entity object** – an object that contains business-related information that is typically persistent and stored in a database.
- Entity objects (instances of entity classes) are used to hold & update information about some phenomenon, such as an event, a person or some real life object. (student, teacher, course etc)
- They represent the system data, often from the domain model.
- Still exists after the use case terminates. (e.g. Database storage)

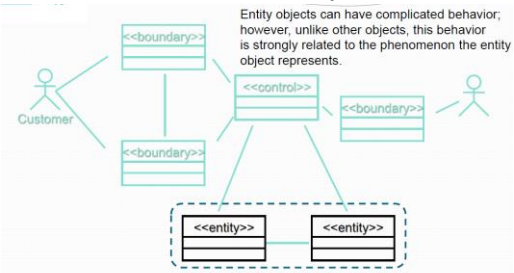
ENTITY OBJECT





Mostly corresponds to conceptual data model classes

Role of an Entity class



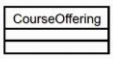
Store and manage information in the system

The values of its attributes and relationships are often given by an actor.
Entity objects are **independent** of the environment (actors)

Examples – Entity class

- Register for Courses (Create Schedule)

A person enrolled in classes at the university



A specific offering for a course including days of week and times

The courses a student has selected for current semester

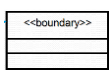
Candidate: Entity class

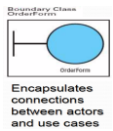
- Sometimes there is a need to model information about an actor within the system. This is not the same as modeling the actors (actors are external by definition). These classes are sometimes called **"surrogates"**.
- For example, a course registration system maintains information about the student 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 nor the student is an actor to the system.

Boundary Class

- **Interface/Boundary object** – an object that provides the means by which an actor can interface with the system.
- Examples include a window, dialogue box, or screen.
- For nonhuman actors, an application program interface (API) is the interface object.
- Boundary class can be either user, system or device interfaces classes.
  - User interface classes facilitate communication with human users of the system.
  - System interface classes facilitate communications with other systems.
  - Device interface classes provides an interface to the devices which detect external events
- One boundary class per use case/per actor

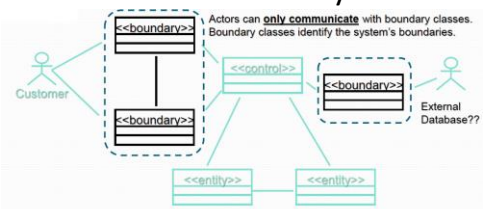
INTERFACE OBJECT





Encapsulates connections between actors and use cases

Role of boundary class



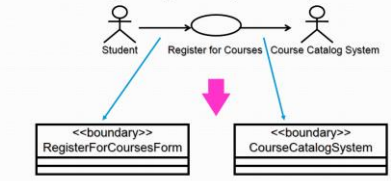
A boundary class is a class used to model interaction between the system's surroundings and its inner workings; Involves transforming and
Boundary Classes model parts of the system that depend on its surroundings. Entity and control classes model parts that are independent of the system's surroundings.
Examples of boundary classes: Classes that handle GUI or communications protocols.

Boundary class

- Identify boundary classes for things mentioned in the flow of events of the use-case realization.
- Consider the **source** for all external events and make sure there is a **way** for the system to detect these events, (user inputs/ responses? Connection to existing external data..)
- **One recommendation:** For the initial definition of boundary classes is **one boundary class per actor/use case pair.**
  - This **class** can be viewed as having the responsibility for coordinating the interaction with the actor.
  - This may be refined as a more detailed analysis is performed.
  - This is particularly true for window-based GUI applications, where there is typically one boundary class for each window, or one for each dialog.

Example: Investigating Boundary class

- One boundary class per actor/use case pair:



•The RegisterForCoursesForm contains a Student's "schedule-in-progress". It displays a list of Course Offerings for the current semester from which the Student may select to be added to his/her Schedule.
•The CourseCatalogSystem interfaces with the legacy system that provides the complete catalog of all courses offered by the university.

Guidelines: Boundary – user interface class

- Concentrate on what information is presented to the user.
- DO NOT concentrate on the UI details.
- Analysis Classes are meant to be a first-cut at the abstraction of the system.
- The boundary classes may be used as 'holding places' for GUI classes.
- Do not do a GUI design in analysis, but isolate all environment-dependent behavior. (Likely you may be able to reverse engineer a GUI component and tailor it.)
- The expansion, refinement and replacement of these boundary classes with actual user interface classes is a very important activity of Class Design.
- If prototyping the interface has been done, these screen dumps or sketches may be associated with a boundary class.
- Only model the key abstractions of the system – not every button, list, etc. in the GUI.

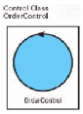
Guidelines: Boundary – System interface

- Concentrate on what protocols must be defined
  - Note that your application must interface with an existing 'information source.'
- Do NOT concentrate on how the protocols will be implemented.
- If the interface to an existing system or device is already well-defined, the boundary class responsibilities should be derived directly from the interface definition.
- If there is a working communication with the external system or device, make note of it for later reference during design.

Control Class

- **Control object** – an object that contains application logic that isn't the responsibility of an entity object.
- **Encapsulates** business functionality
- Proposed in Rational Unified Process (RUP)
- Examples of such logic are business rules and calculations that involve multiple objects
- Control objects coordinate messages between interface objects and entity objects and the sequences in which the messages occur.

CONTROL OBJECT



Mostly performs behaviors associated with inner workings of use cases

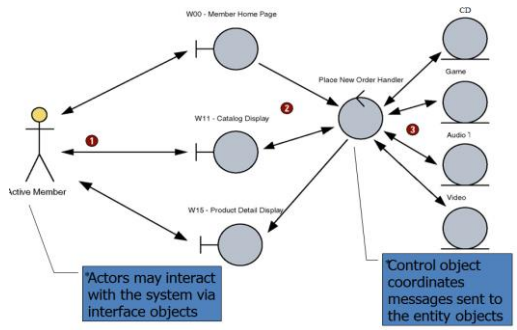
Analysis Classes

- Entity class:
  - Persistent data
  - Used multiple times and in many UCs
  - Still exists after the UC terminates (e.g. DB storage)
- Boundary class:
  - (User) interface between actors and the system
  - E.g. a Form, a Window (Pane)
- Control class:
  - Encapulates business functionality
  - Proposed in RUP (Rational Unified Process)

Identify and Classify Use-Case Design Objects

INTERFACE OBJECTS	CONTROLLER OBJECTS	ENTITY OBJECTS
W00 – Member Home Page	Place New Order Handler	Billing Address
W02 – Member Profile Display		Shipping Address
W03 – Display Order Summary		Email Address
W04 – Display Order Confirmation		Active Member
W09 – Member Account Status Display		Member Order
W11 – Catalog Display		Member Ordered Product
W15 – Product Detail Display		Product
		Title
		Audio Title
		Game Title
		Video Title
		Transaction

Model High-Level Interactions with Object Robustness Diagrams



Rules for Robust analysis: ECB Pattern

- Structural restrictions for analysis classes:
  - Entity: only attributes (+ get/set/find methods)
  - Control: only methods: (at least) one method / UC
  - Boundary: both attributes and methods.
- Relationship between analysis classes (Layers)
  - One boundary class for each Actor – UC relation.
  - Entities are only accessed by control objects.
  - Control objects may communicate with all entities, boundaries and control objects.

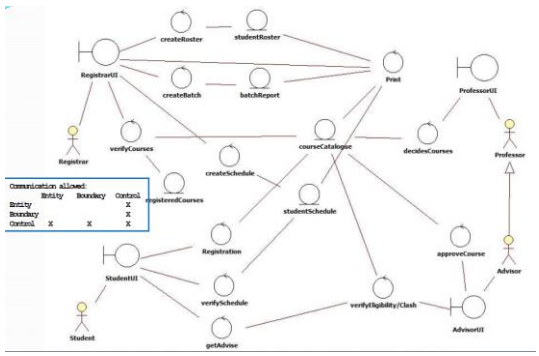
Example - Case

- The UNIVERSITY OF KARACHI registration is briefly described as under:
  - You have been asked to streamline, improve and automate the process of assigning professors to courses and the registration of students such that it takes advantage of prevailing web technologies for on-line real time, location independent access.
  - The process begins by professors deciding on which courses they will teach for the semester. The Registrar's office then enters the information into the computer system, allocating times, room and student population restrictions to each course. A batch report is then printed for the professors to indicate which courses they will teach. A course catalogue is also printed for distribution to students.
  - Students then select what courses they desire to take and indicate these by completing paper-based course advising forms. They then meet with an academic advisor who verifies their eligibility to take the selected courses, that the section of courses are still opened and that the schedule of courses does not clash.

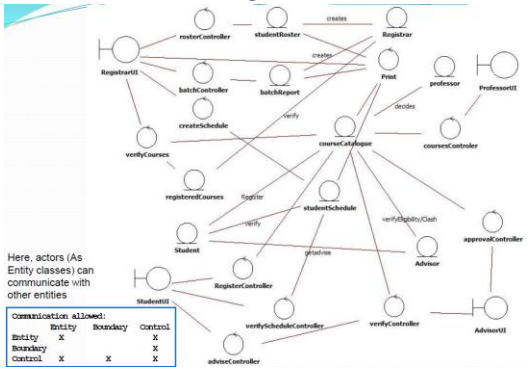
Case

- The typical student load is four courses.
- The advisor then approves the courses and completes the courses registration forms of the student. These are then sent to the registrar who keys them into the registration system – thereby formally registering a student.
- If courses selected are not approved, the student has to select the other courses and completes a fresh course advising form.
- Most times students get their first choice, however in those cases where there is a conflict, the advising office talks with the students to get additional choices.
- Once all students have been successfully registered, a hard copy of the student's schedule is sent to the students foe verification.
- Most students registrations are processed within a week, but some exceptional cases take up to two weeks to resolve.
- Once the initial registration period is over, professors receive a student roster for each class they are schedule to teach.

Robustness Diagram – ECB Pattern



Robustness Diagram – ECB Pattern



END OF TOPIC 10

-COMING UP!!!!!!

-RUP
-Sequence Diagrams
-Collaboration Diagrams