Analysis Model

CSIT 315

H. Johnson

Analysis model (AM)

- The first step in describing ***how*** the system will implement the requirements specification
- Input artifacts:
  - Detailed use cases
  - Use case model
  - User experience model
  - Business entities model

AM Structure

- Key Abstraction folder
  - For each business package
    - Boundary classes
    - Control classes
    - Entity classes
- Use Case Realization folder
  - For each business package
    - For each use case
      - A VOPC diagram
      - For each major flow of the use case
        - A sequence diagram

Comments on previous slide

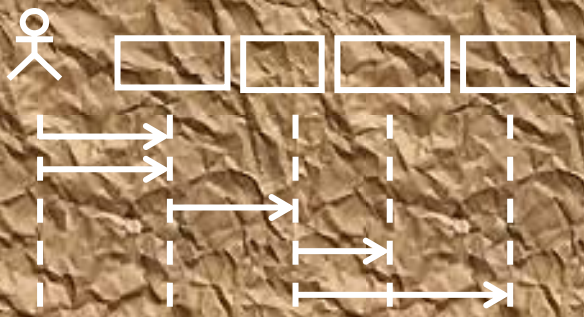
- A use-case realization describes how a particular use case is realized within the Design Model, in terms of collaborating objects.
  - A use-case realization is **one possible realization** of a use case.
  - A use-case realization in the Design Model can be ***traced*** to a use case in the Use-Case Model.
  - A realization relationship is drawn from the use-case realization to the use case it realizes.
- A use-case realization can be represented using a set of diagrams (the number and type may vary)
- Different organizations may have their own way of doing these things…(and naming them…)

Review: Use-Case Realization

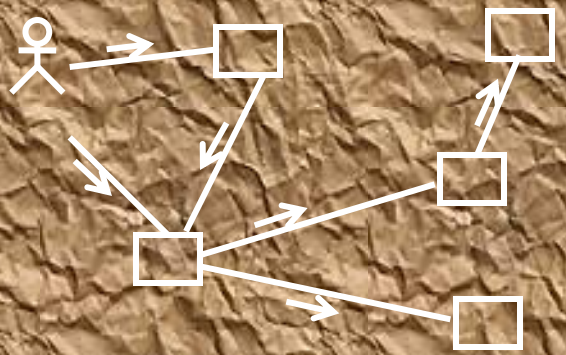
Use-Case Model *Design Model*

Use Case

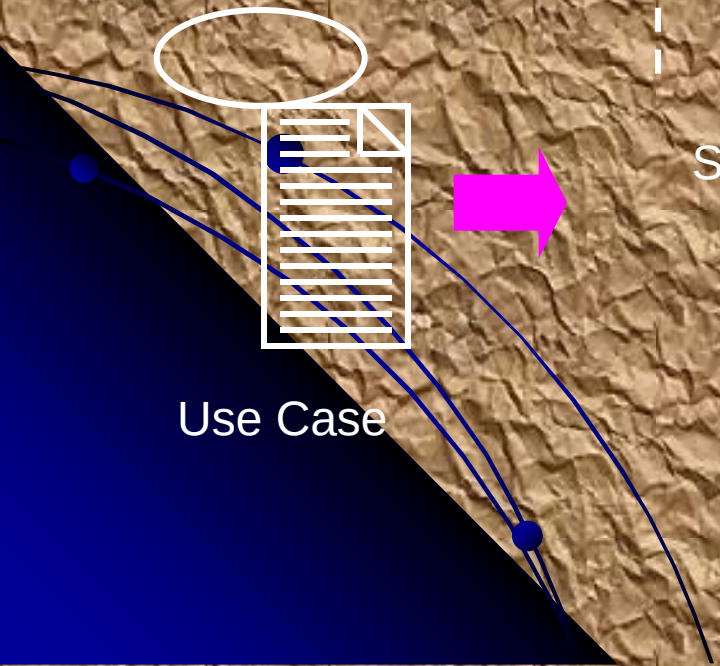
Use-Case Realization



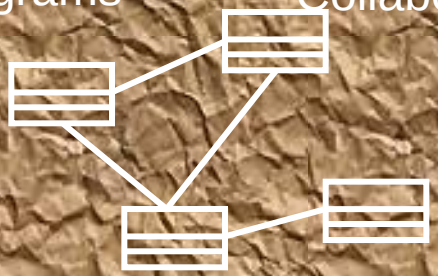
Sequence Diagrams



Collaboration Diagrams



Use Case



Class Diagrams

Not what we have so far.
These are Design Classes. More later

Continuing Comments…

- The diagrams that may be used to realize a use case realization may include:
  - **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.
  - 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.

- A designer: responsible for the integrity of the use-case realization.
- Must coordinate with the designers responsible for the classes and relationships employed in the use-case realization.
- The use-case realization can be used by class designers to understand the class's role in the use case and how the class interacts with other classes.
  - This implies that a team will/may distribute responsibilities for each class to developers.
- This information can be used to determine/refine the class responsibilities and interfaces.
- Find the classes from different behaviors the classes must provide…

Analysis Model Structure

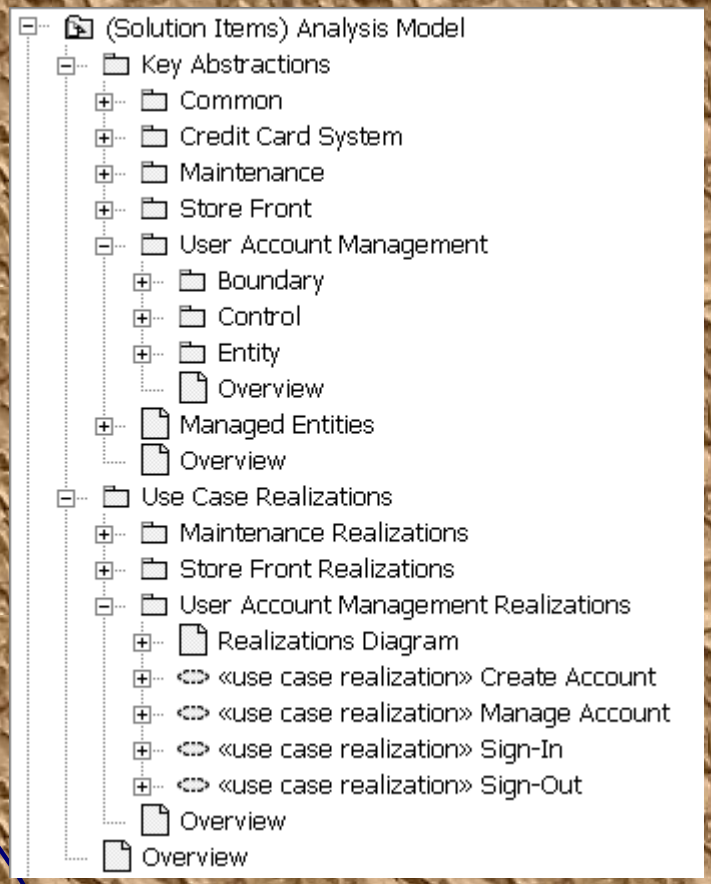


Figure 5-1

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.

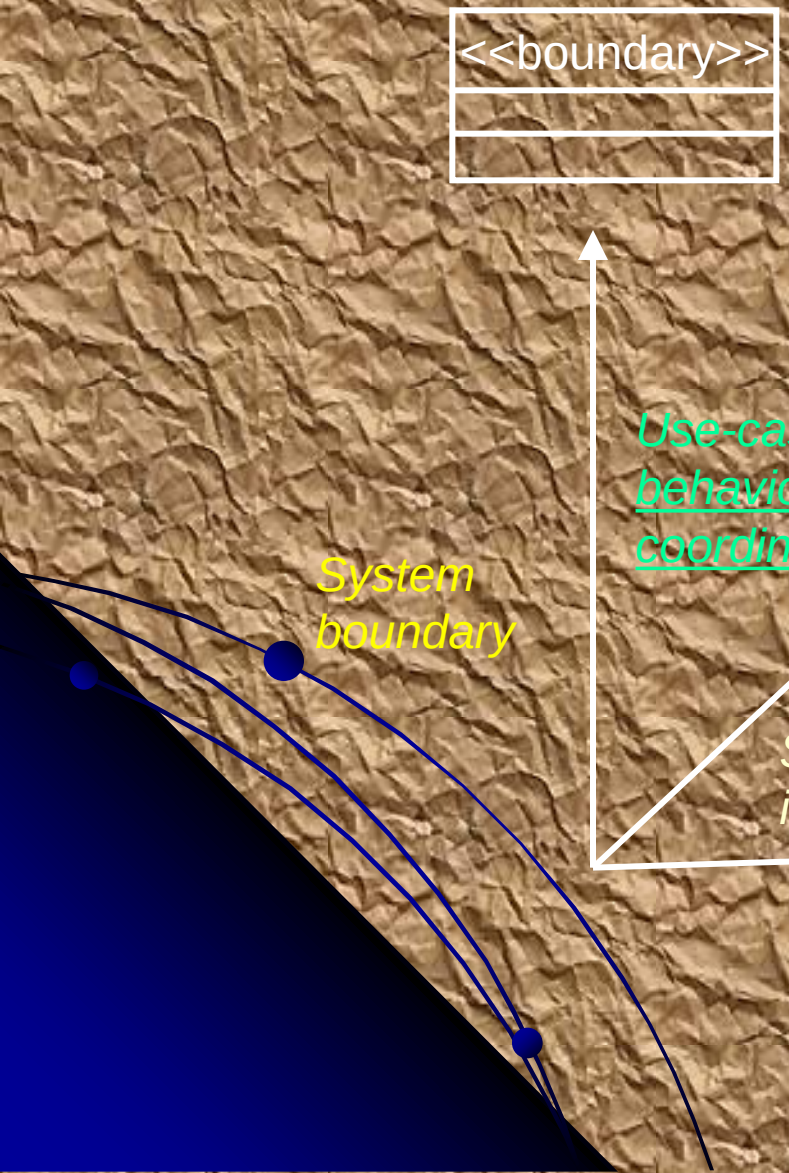
Kinds of Analysis Classes

- 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
  - Each of these has a set of canned duties & responsibilities
  - Again, the distinction between these classes is used in analysis but goes away in design or becomes less of an issue, as we transition these analysis classes into design artifacts / design entities to accommodate the problem domain representations in a solutions space.

What is an Analysis Class?

Can use with the name of the stereotype In guillemets or as symbols with unique icons.

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.



Boundary, Control, & Entity

Figure 5-3

Analysis Classes – an Early Conceptual Model

- The analysis classes, taken together, represent an early conceptual model of the system.
- This conceptual model evolves quickly and remains fluid for some time as different representations and their implications are explored.
- Don’t spend a lot of time maintaining this model, as this ‘model’ is largely expendable.
- Analysis classes rarely survive into the design unchanged.
- Many of them represent whole collaborations of objects, often encapsulated by a single subsystems or a reverse engineered component.

Analysis Classes – Early Conceptual Model

- Analysis classes are 'proto-classes', which are essentially "clumps of behavior".
- Analysis classes are early conjectures of the composition of the system; they rarely survive intact into implementation.
- Many of the analysis classes morph into something else later on (subsystems, components, split classes, combined classes).
- They provide us with a way of capturing the required behaviors in a form that we can use to explore the behavior and composition of the system.
- Analysis classes allow us to "play" with the distribution of responsibilities, re-allocating, as necessary.

Model Structure

Tracing Use Case Realizations to Use Cases & Ux storyboards

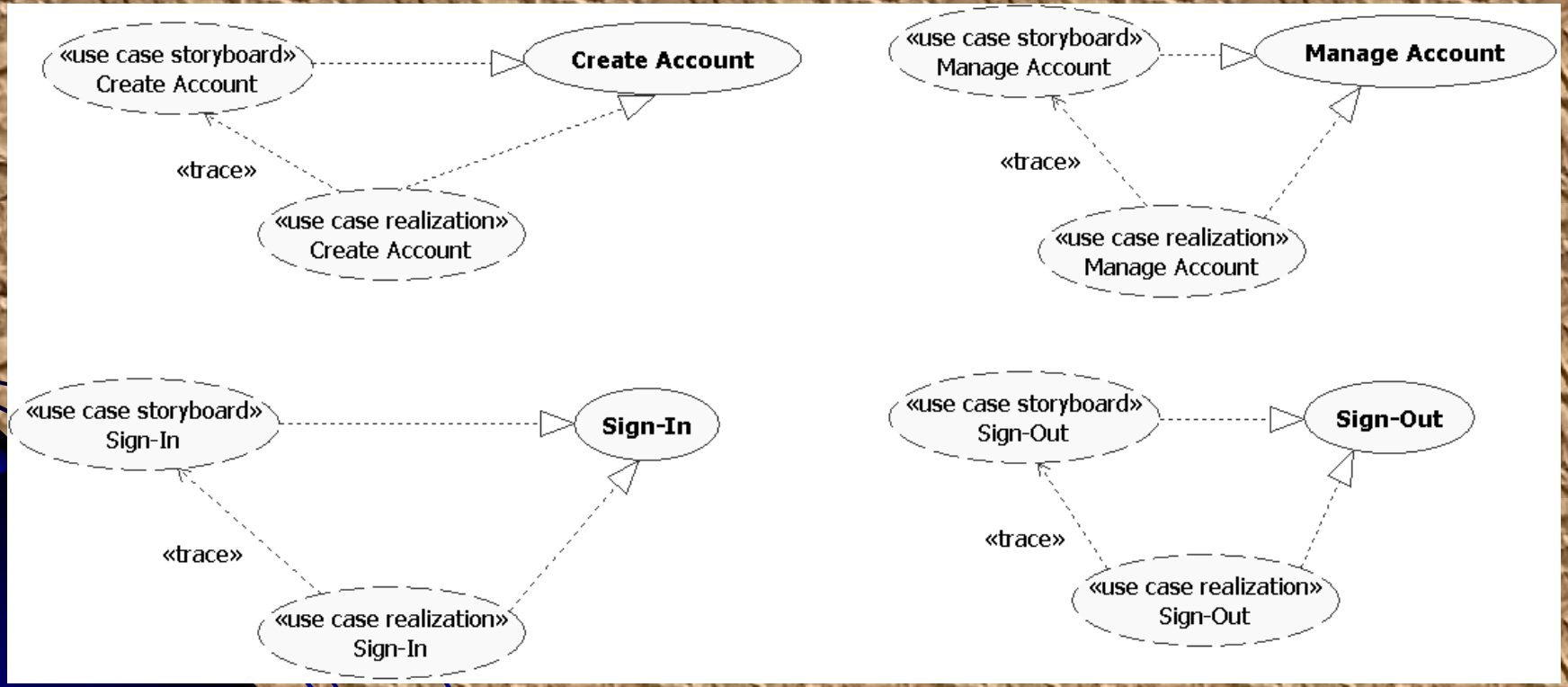


Figure 5-2

AM: Key Abstractions

- Boundary classes
  - Model the interaction of the system with its actors
    - For human actors – maps to web pages/HTML forms
    - For external systems – maps to adapters
- Control classes
  - Capture the control logic (presentation and business logic) of one or more use cases
- Entity classes
  - Represent the business concepts manipulated by the system
- No attributes or operations are specified for key abstraction classes in analysis model

AM: Boundary classes

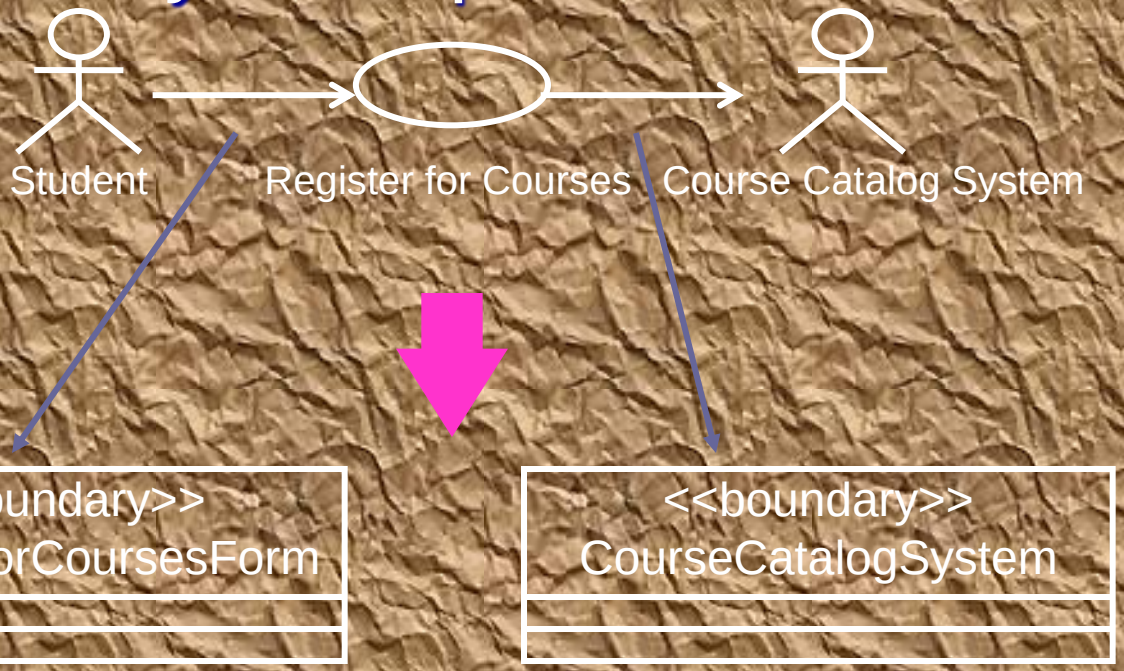
- Model the interaction of the system with its actors (Human actors and External systems)
- Actors only interact with boundary classes
- Boundary classes may interact with other boundary classes, actors, and control classes
- *Create a boundary class for each screen in the user experience model – one-to-one relationship between boundary classes and the screens*
- *Create a boundary class for each external system*
- *No boundary class for input forms*

The Role of a Boundary Class

- A boundary class is a class used to model interaction between the system's surrounding and its inner workings; Involves transforming and translating events and noting change in the system presentation (such as the interface).
- 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.

Example: Finding Boundary Classes

- 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 unabridged catalog of all courses offered by the university.

Guidelines: Boundary Class – User Interface classes

- User Interface Classes
  - 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.
  - Boundary classes may be used as 'holding places' for GUI classes. (Addressed in much more detail later)
  - 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 – later
  - 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.

Tracing boundary classes to screens for the User Account Management package

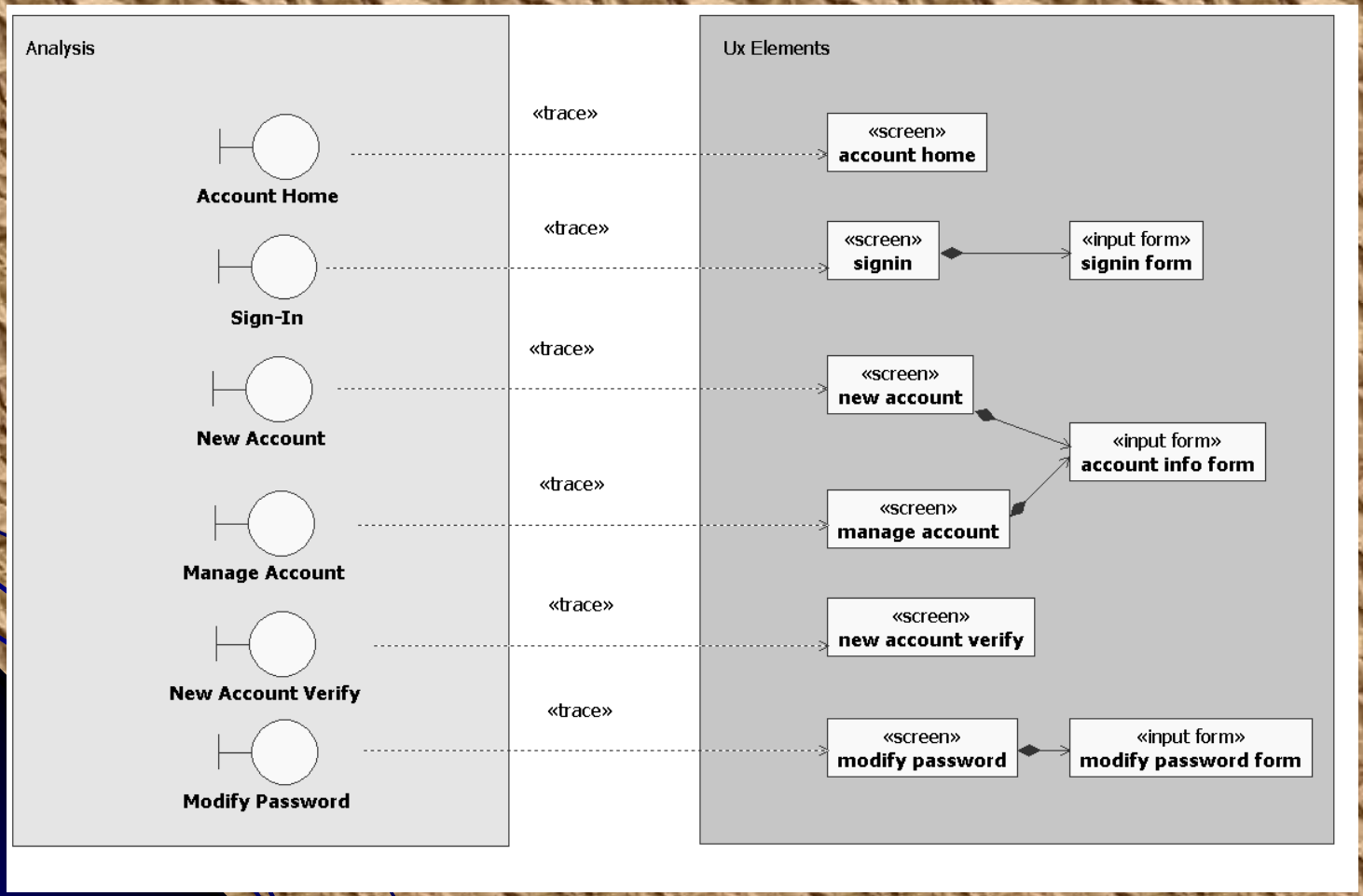


Figure 5-4

Use case model: The Create Account Use Case Perspective

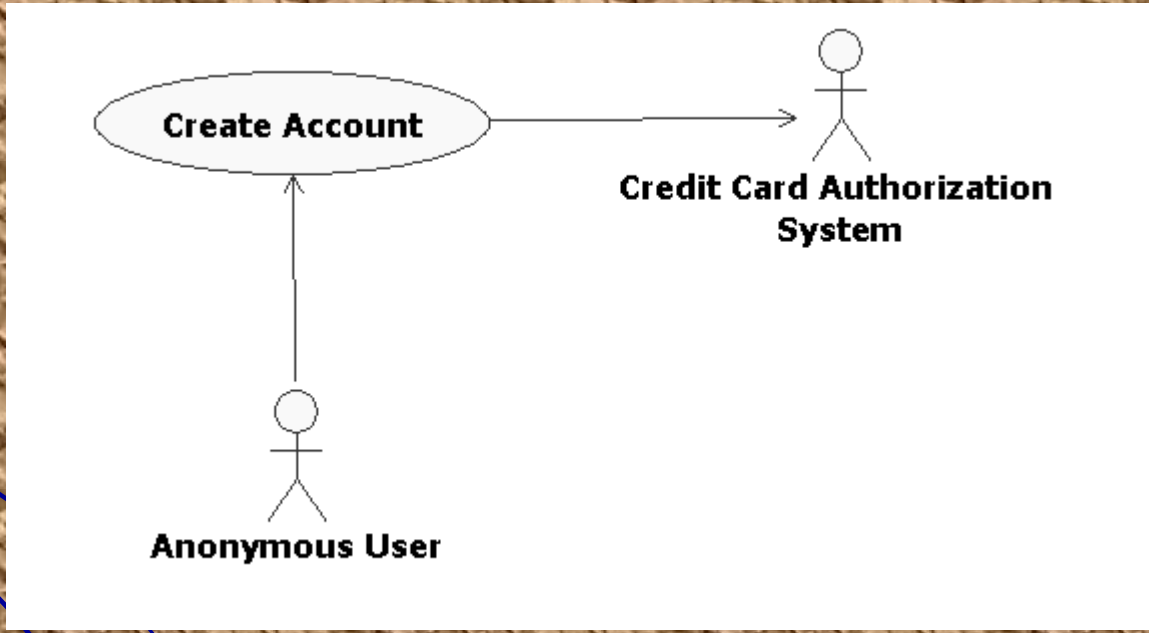
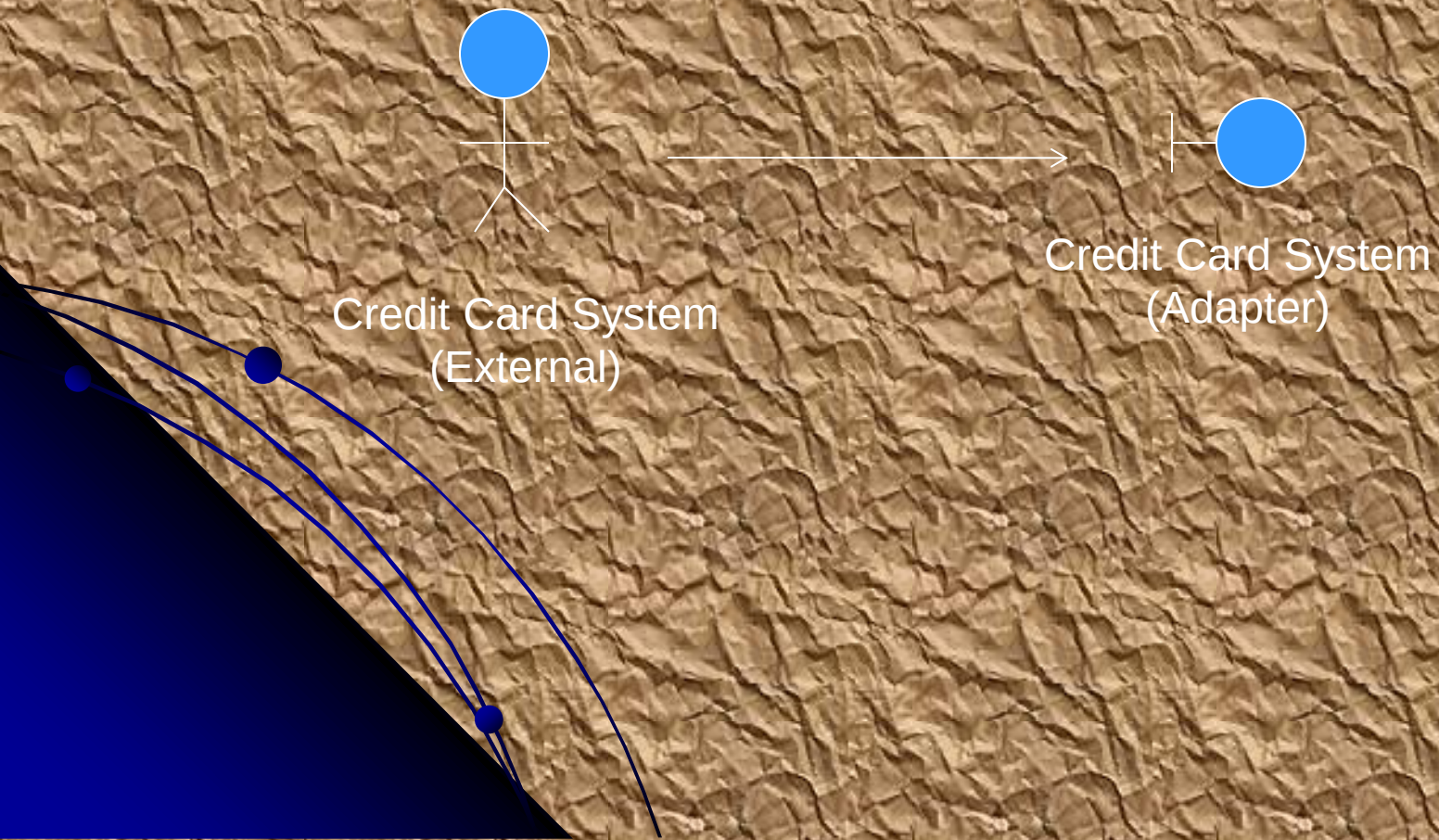


Figure 5-11

Boundary class for Credit Card External System



AM: Control Classes - 1

- Capture the control logic of one or more use cases
- Presentation logic
  - *Dispatcher* –control the flow of execution for a specific use case
  - Its responsibilities are assigned based on the analysis of the flow of events in the use case
- Business logic
  - *Entity manager* – control the access to one or more related entity classes for a specific use case package, defining a business interface to those classes.
  - Its responsibilities are assigned based on what entities are involved in the use case and what their roles are.

AM: Control classes - 2

- *Dispatcher classes:*
  - *Define a dispatcher control class for each use case*
  - *e.g., CreateAccountDispatcher for the Create_Account use case*
- *Entity Manager classes:*
  - *Create an entity manager control class for each use case package.*
  - *e.g., AccountManager for the Account_Management use case package*
- *Merge dispatcher control classes as appropriate.*

Associations between control classes in the User Account Management package

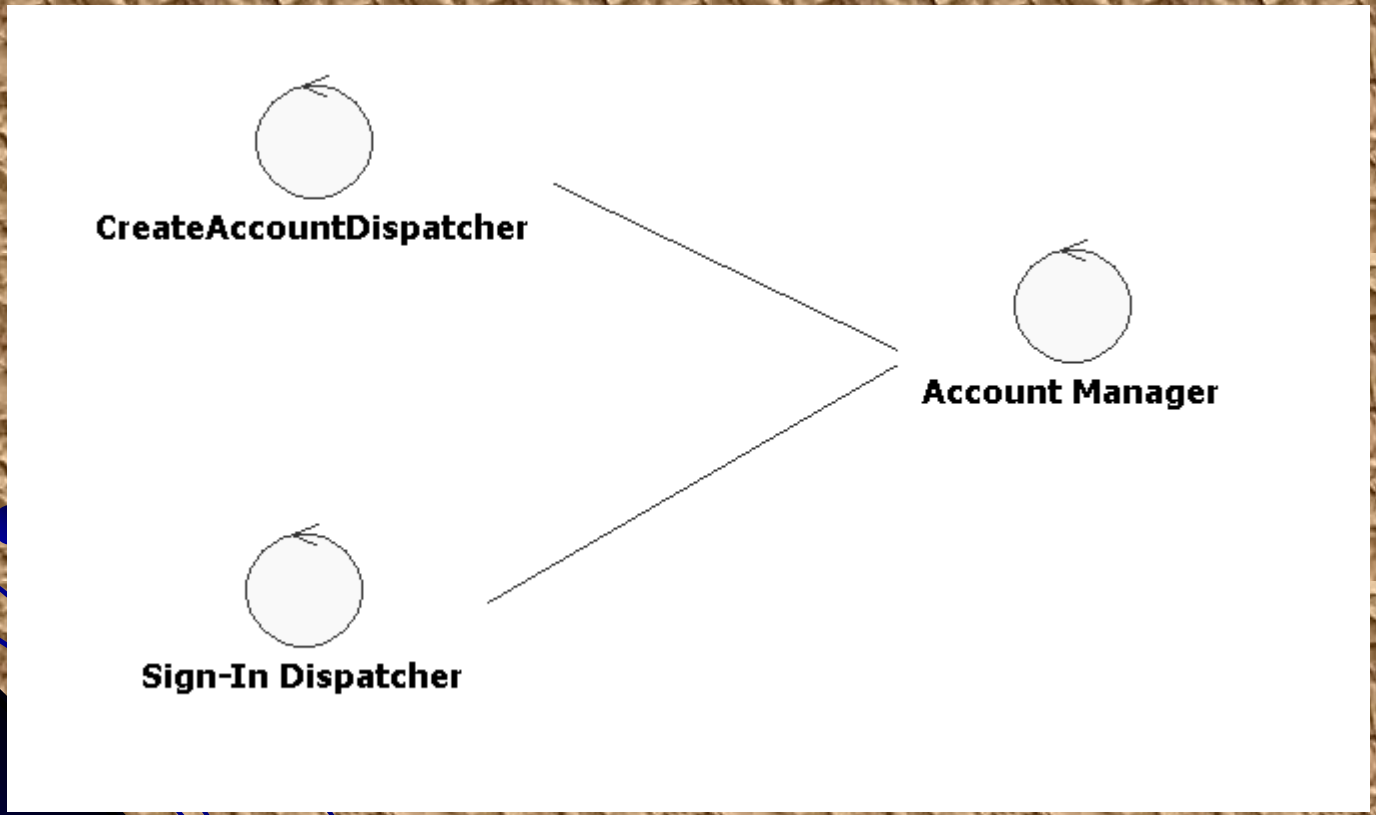


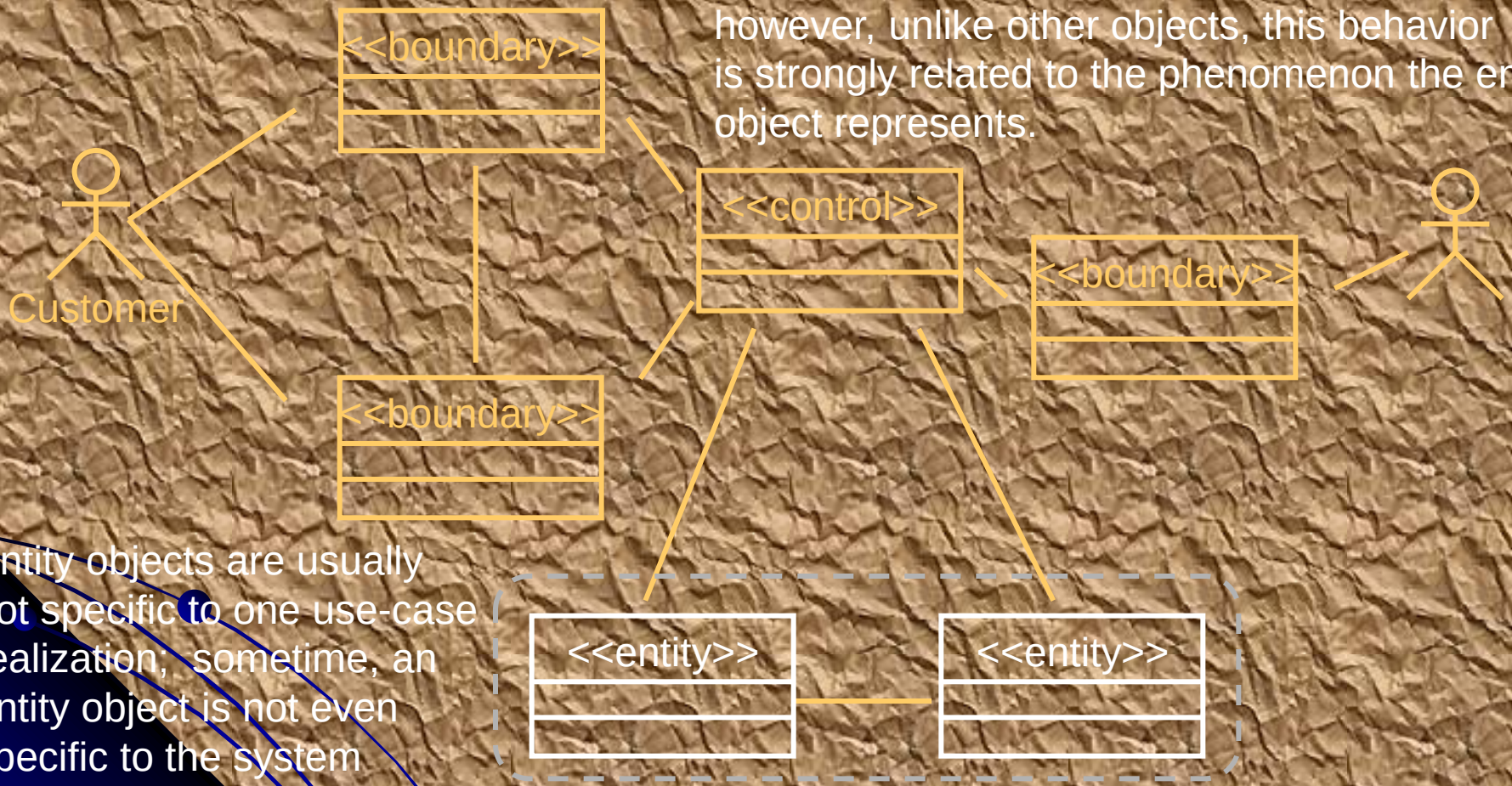
Figure 5-5

Entity Classes

- Entity classes represent stores of information in the system
- They are typically used to represent the key concepts the system manages.
- Entity objects (instances of entity classes) are used to hold and update information about some phenomenon, such as an event, a person, or some real-life object.
  - (Chapter advisor, memorabilia, university, student, correspondence_item, International_Secretary, …)
- They are usually persistent, having attributes and relationships needed for a long period, sometimes for the life of the system.
- **The main responsibilities of entity classes are to store and manage information in the system.**

The Role of an Entity Class

Entity objects can have complicated behavior; however, unlike other objects, this behavior is strongly related to the phenomenon the entity object represents.



Entity objects are usually not specific to one use-case realization; sometime, an entity object is not even specific to the system itself.

Store and manage information in the system

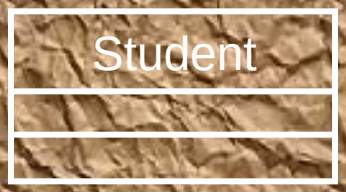
The values of its attributes and relationships are often given by an actor.
Entity objects are **independent** of the environment (actors)

Example: Finding Entity Classes

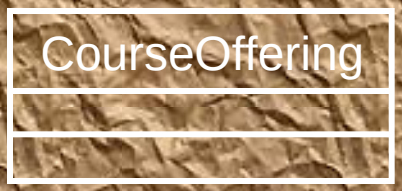
- Use use-case flow of events as input
- Key abstractions of the use case
- Traditional, filtering nouns approach
  - Underline noun clauses in the use-case flow of events
  - Remove redundant candidates
  - Remove vague candidates
  - Remove actors (out of scope)
  - Remove implementation constructs
  - Remove attributes (save for later)
  - Remove operations

Example: Candidate Entity Classes

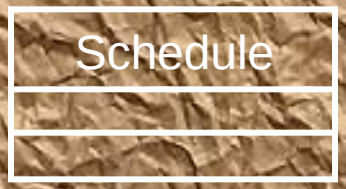
- Register for Courses (Create Schedule)



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 Classes - continued

- 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.

AM: Entity classes - 1

- Represent the business concepts manipulated by the system
- Normally entity classes belong to the whole system -- enterprise.
- The basis for the logical design of the database
- All communication to entity classes should go through an entity manager
- *Derive entity classes from the business entity model*
- *Package entity managers are responsible for the management of the entities identified for the use cases in the package. (e.g., AccountManager is responsible for CreditCard, UserAccount and UserGroup entity classes for the UserAccountManagement use case package)*

AM: Entity classes - 2

- How to identify entity classes
  - Noun identification technique:
    - *For each use case, underline nouns and noun phrases*
    - *Eliminate duplicates and unsuitable ones to have a draft of the class list*
  - Example:
    - *USE CASE: A student selects a course and registers in one of its sections provided the student meets all the prerequisites of the course. A professor may select a section of a course to teach and obtain a student list for the section.*
    - Identified Classes: *student, course, section, and professor*

AM: Entity classes - 3

- Three types of associations between entity classes:
  - *Unspecified association*: for classes managed by two separate managers
  - *Aggregation associations*: one class is part of another and the part class may belong to more than one aggregate
  - *Composition associations*: like aggregation, but part classes cannot belong to other classes, the parts live and die with the whole.

Entity model for the User Account Management package

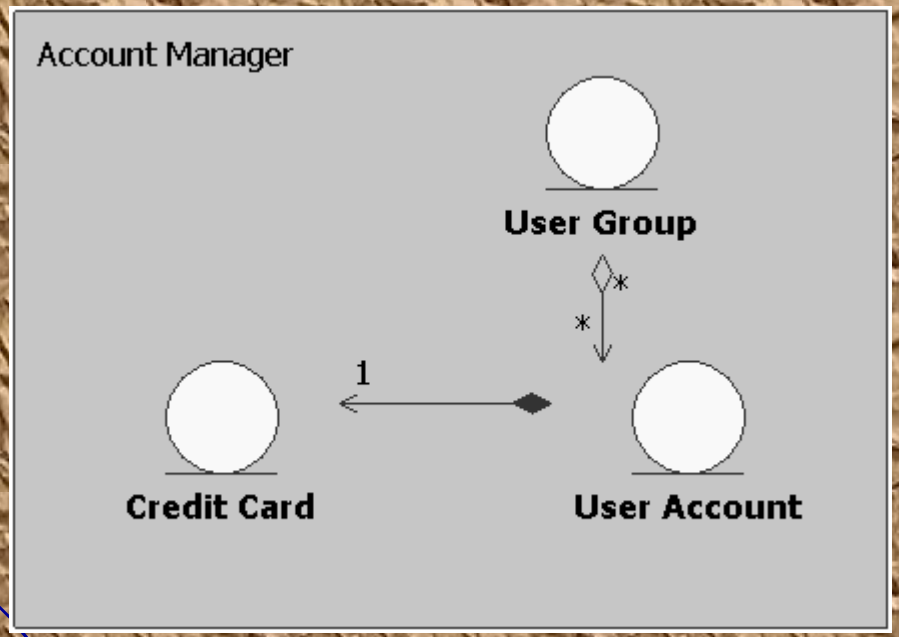


Figure 5-7

Managed Entities diagram

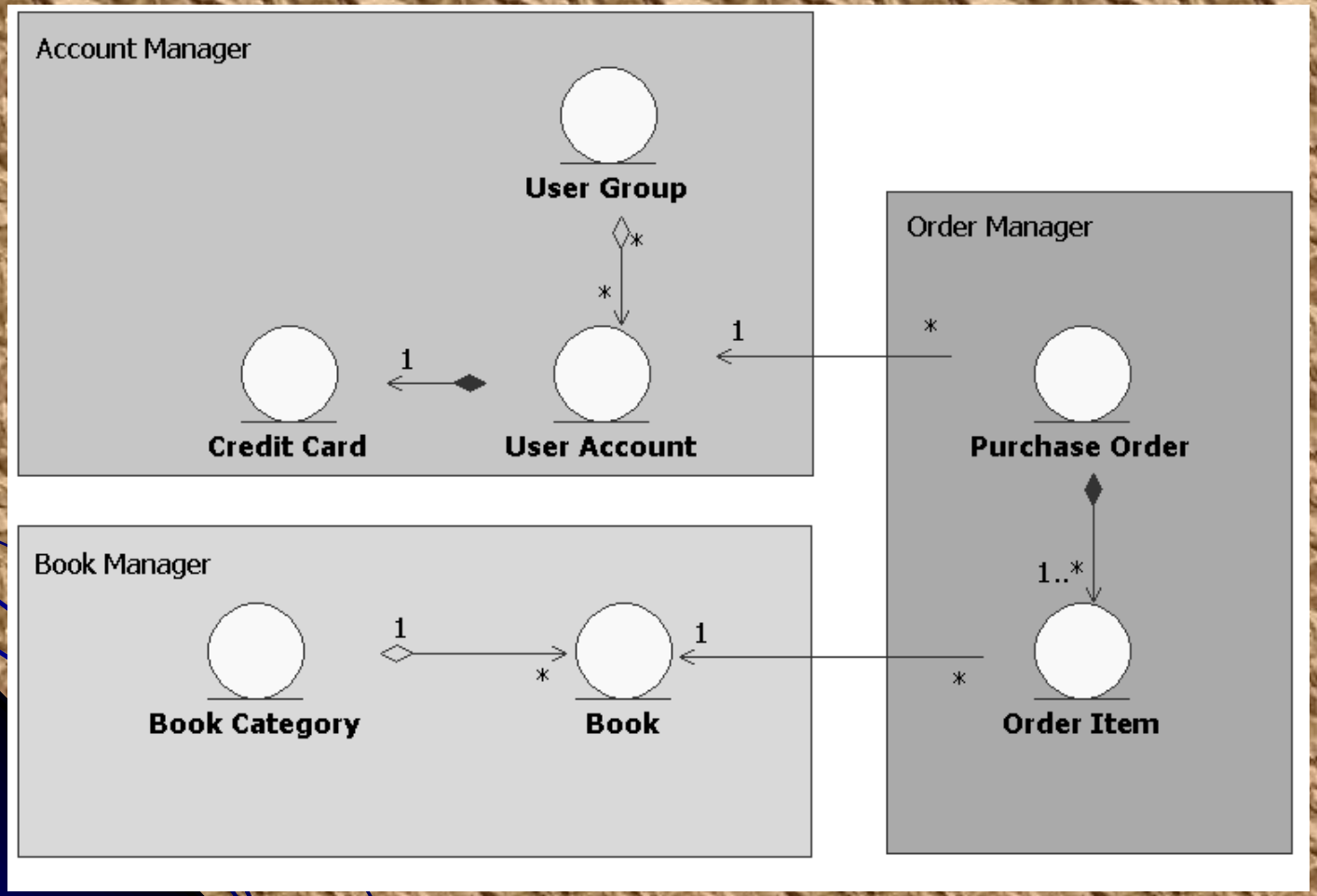
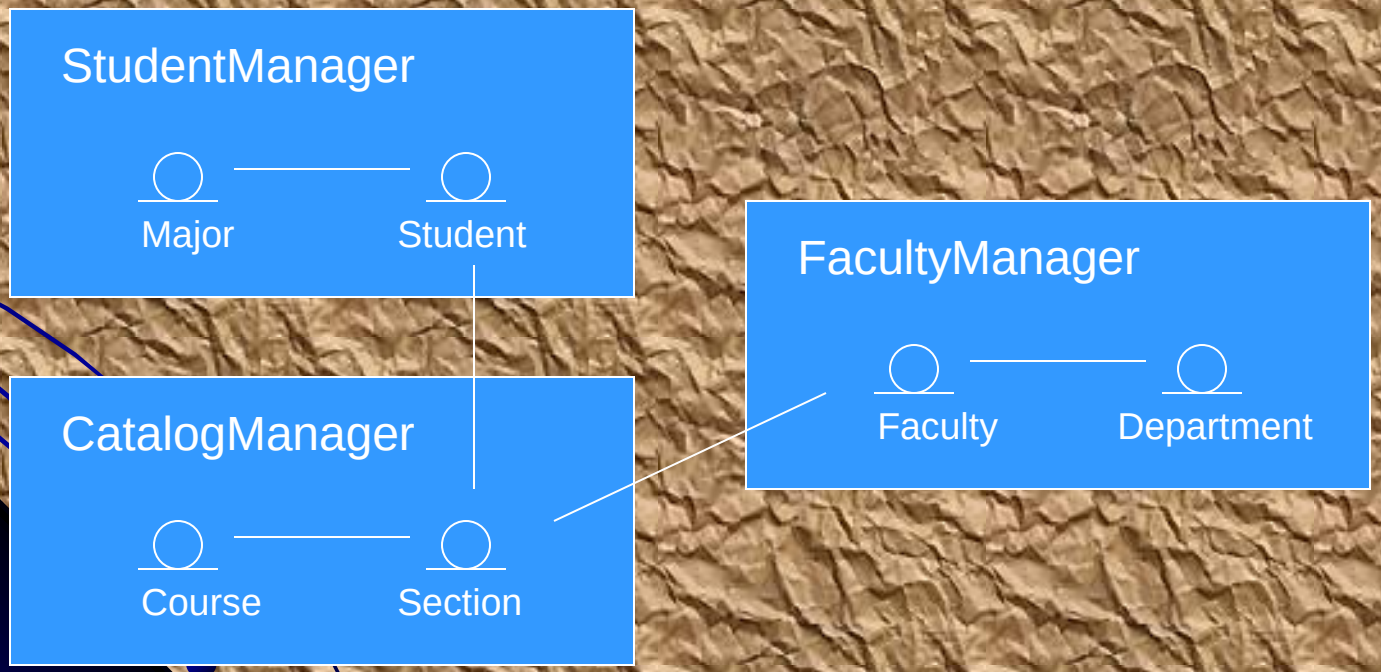


Figure 5-8

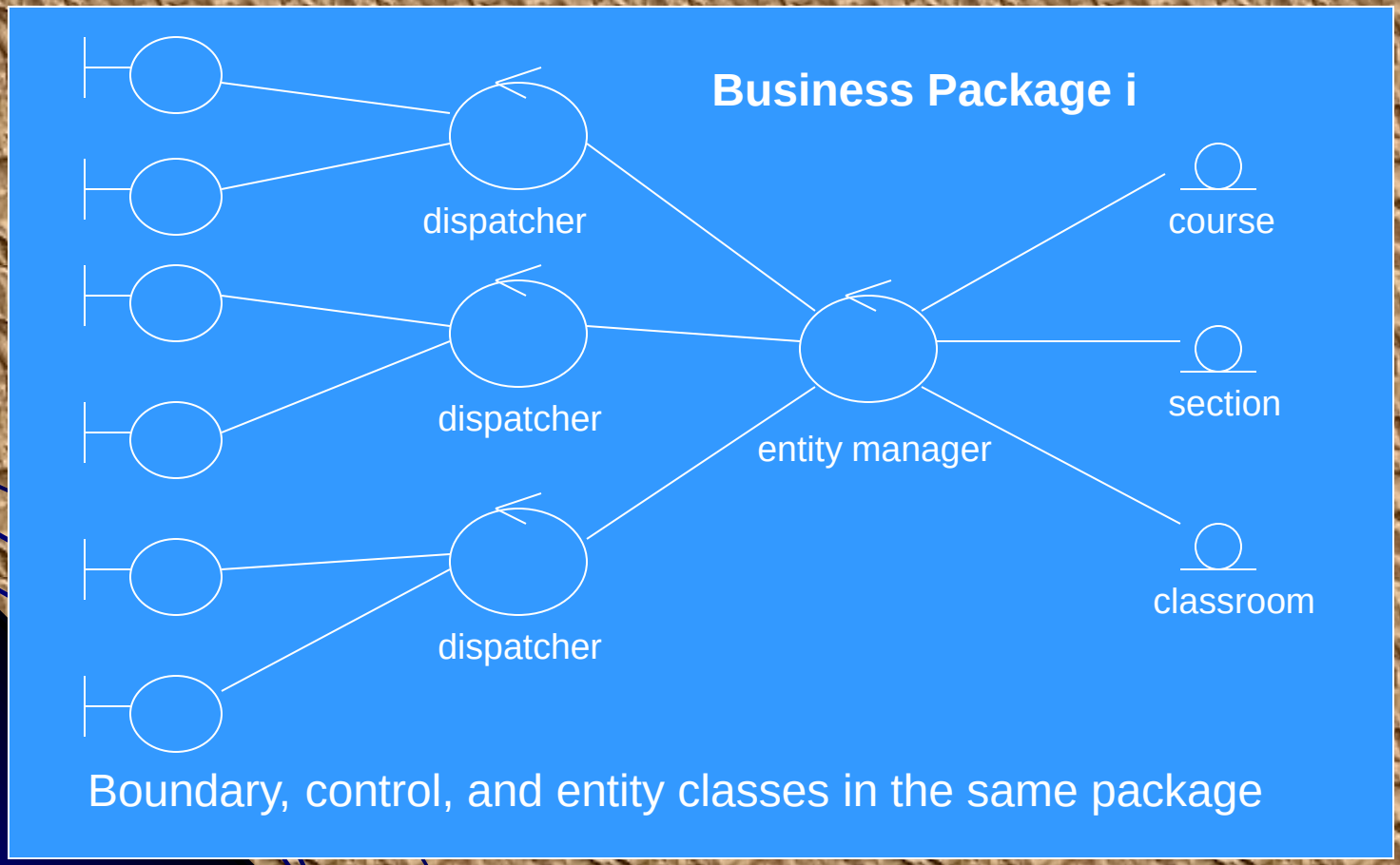
AM: Managed entities diagram

- Shows the manager of each package and the entities managed by each manager and the relationship between entities



Note: For simplicity, cardinality ratios are not shown here.

AM: Relationship between classes



Boundary, control, and entity classes in the same package

AM: VOPC diagram

- VOPC: View Of Participating Classes
- Show how the system objects work together in the realization of a use case.
- Use key abstraction classes
- Show all the objects involved in a use case for the basic flow and alternate flows as well.
- Use simple, non-directed associations between classes involved.
- Avoid too much details
- One VOPC diagram per use case

VOPC diagram for the Create Account use case

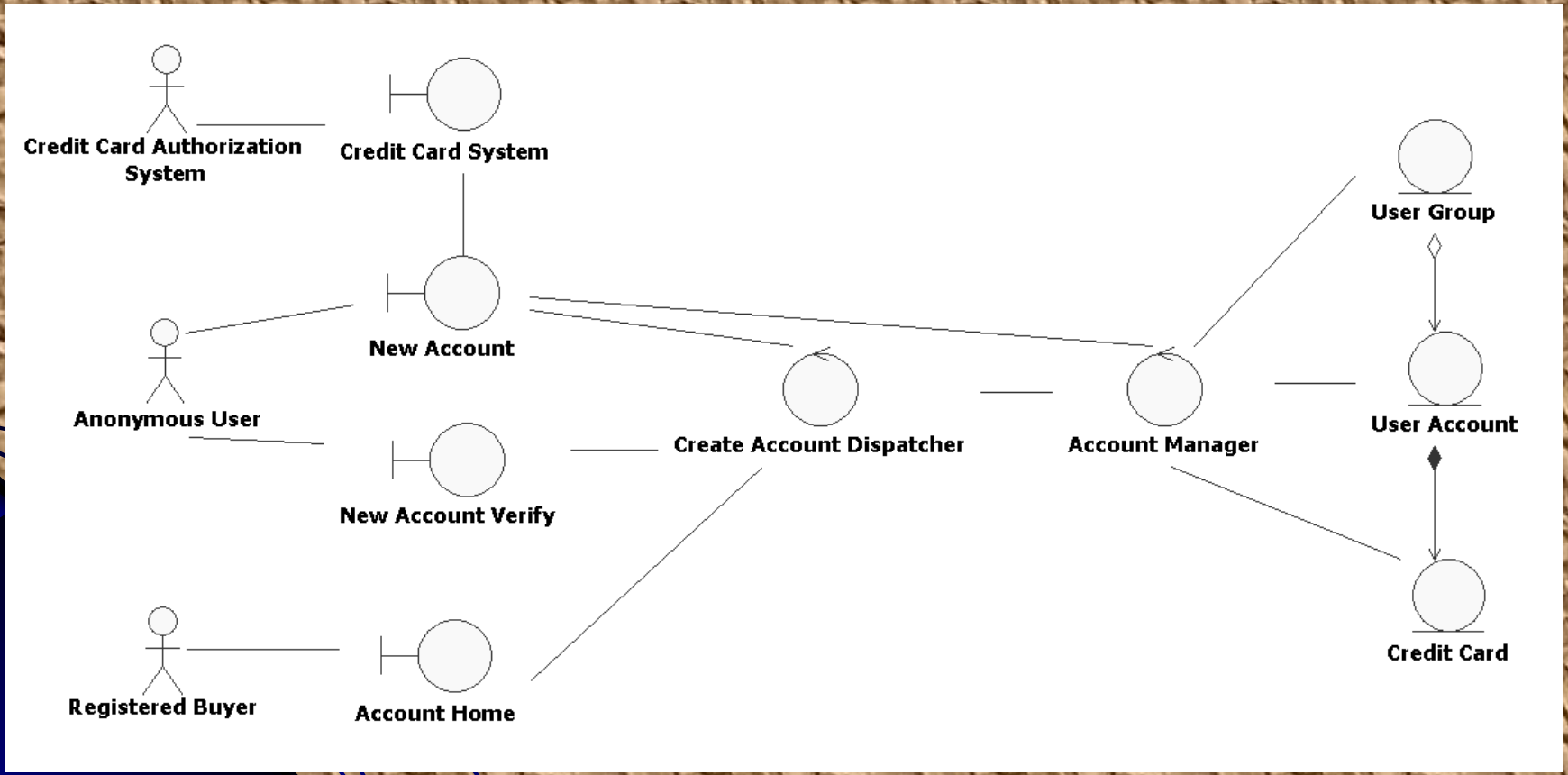


Figure 5-9

AM: Sequence diagram I

- The VOPC diagram for the use case is the primary source of objects for the sequence diagram
- Represent a step-by-step description of a complete path through one scenario of the use case.
- Create at least one sequence diagram for the basic flow of each use case
- The messages should be descriptive since they normally do not match operations of the associated classes.
- One sequence diagram for each possible use case scenario of interest.

AM: Sequence diagram II

- Re-write use case description using the boundary, control, and entity classes.
- Each step describes “noun-verb-noun”
  - Nouns: actors, boundary, entity classes
  - Verbs: the action the 1st noun asks the 2nd noun to take.
- Attach the description of each step to the sequence diagrams.

Sequence diagram for the Create Account use case

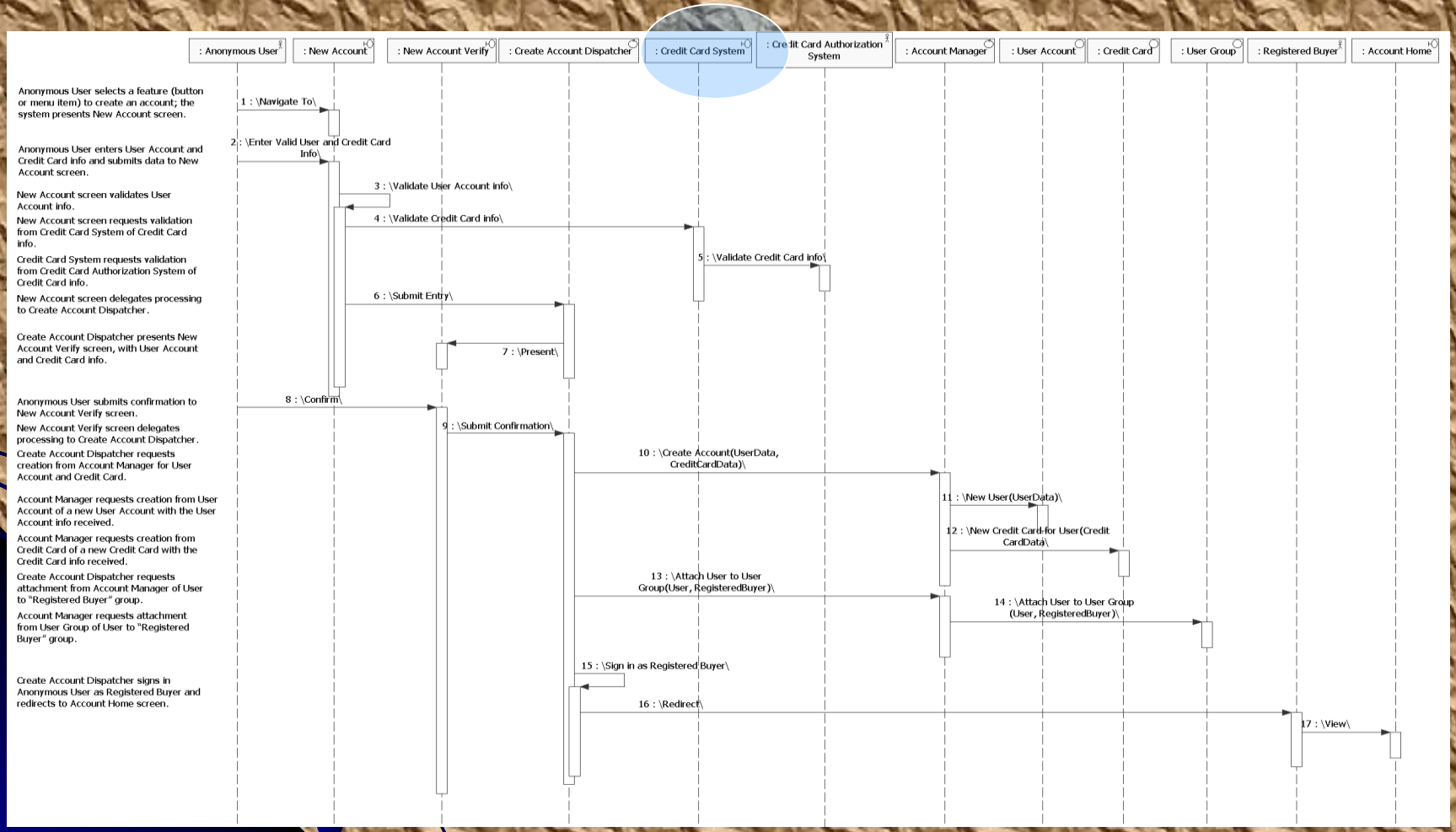


Figure 5-10

An alternate collaboration when signing in Anonymous User

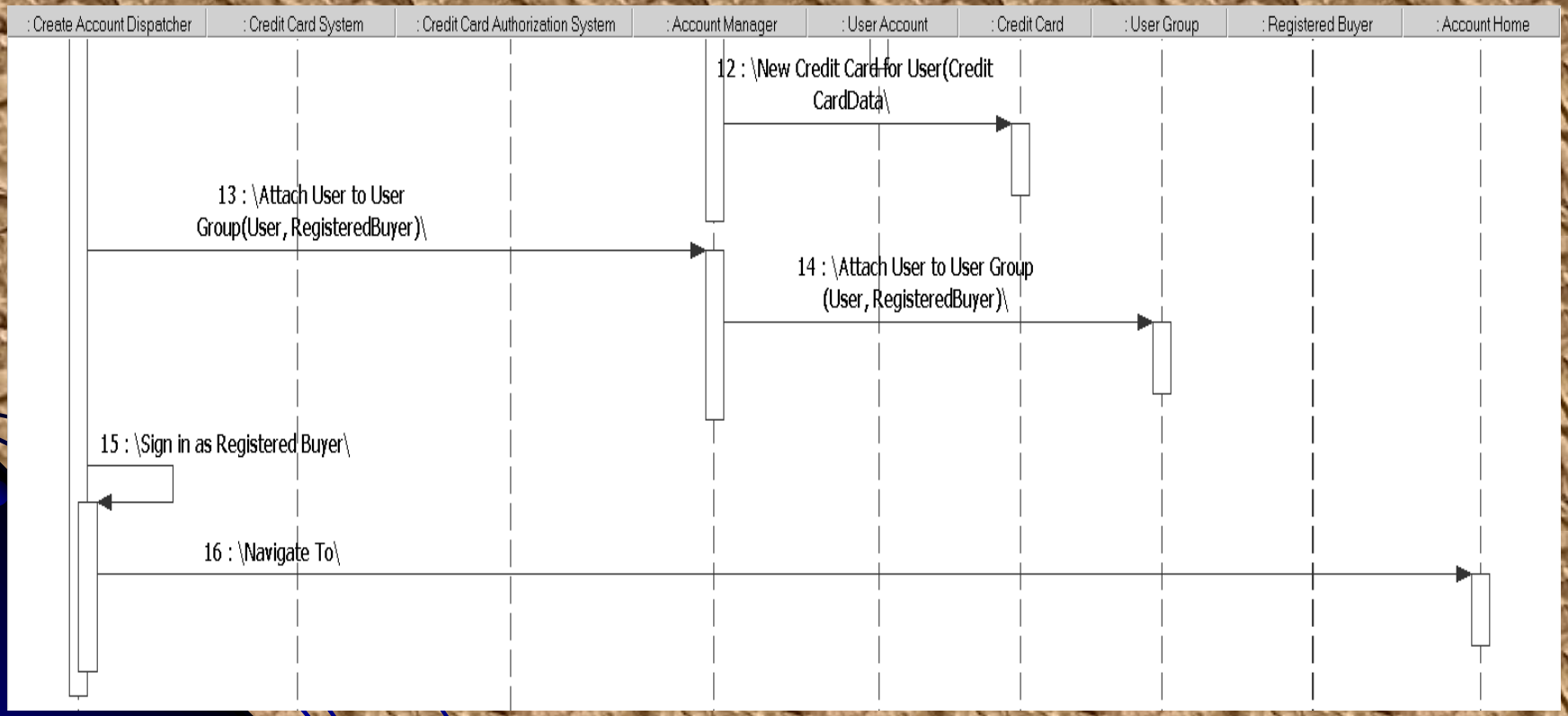


Figure 5-12

AM: Define system tests

- Use use-case scenarios to define test cases.
- Consider all possible combinations of flows of events
- Combine one or more flows of events into a use case scenario.

AM: Summary

- Three key Abstraction classes:
  - Boundary classes
    - One screen in Ux is mapped to a boundary class
    - Each external system is assigned a boundary class
  - Control classes
    - A dispatcher control class per use case
    - An entity manger control class per use case package
  - Entity classes
    - Each business concept in the business entity model is mapped to an entity class.
- Managed Entities diagram
  - An entity manager controls the access to the entities in the use case package
  - Represents all the persistent entities of the system
- VOPC diagram
  - One VOPC per use case
- Sequence diagram
  - One sequence diagram per use case scenario of interest
  - At least one sequence diagram for the basic flow of the use case