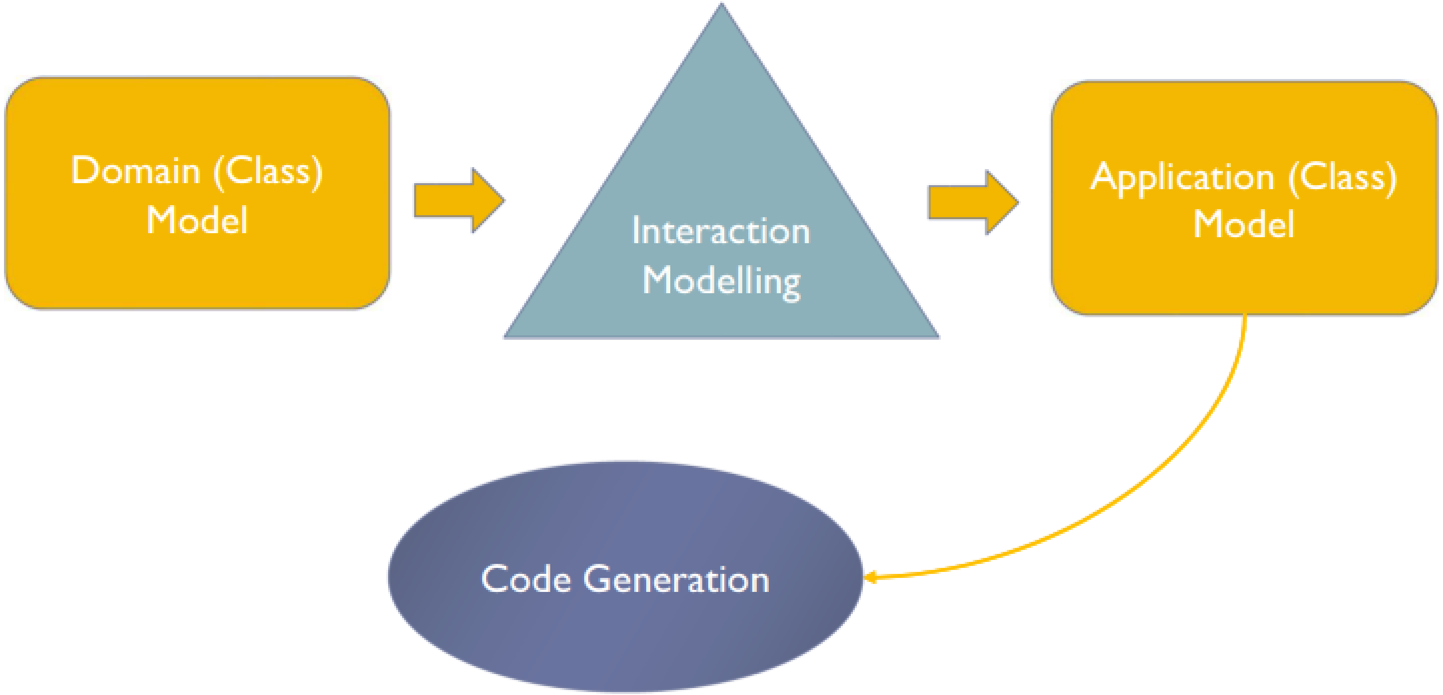
Interaction Modelling: Sequence Diagrams

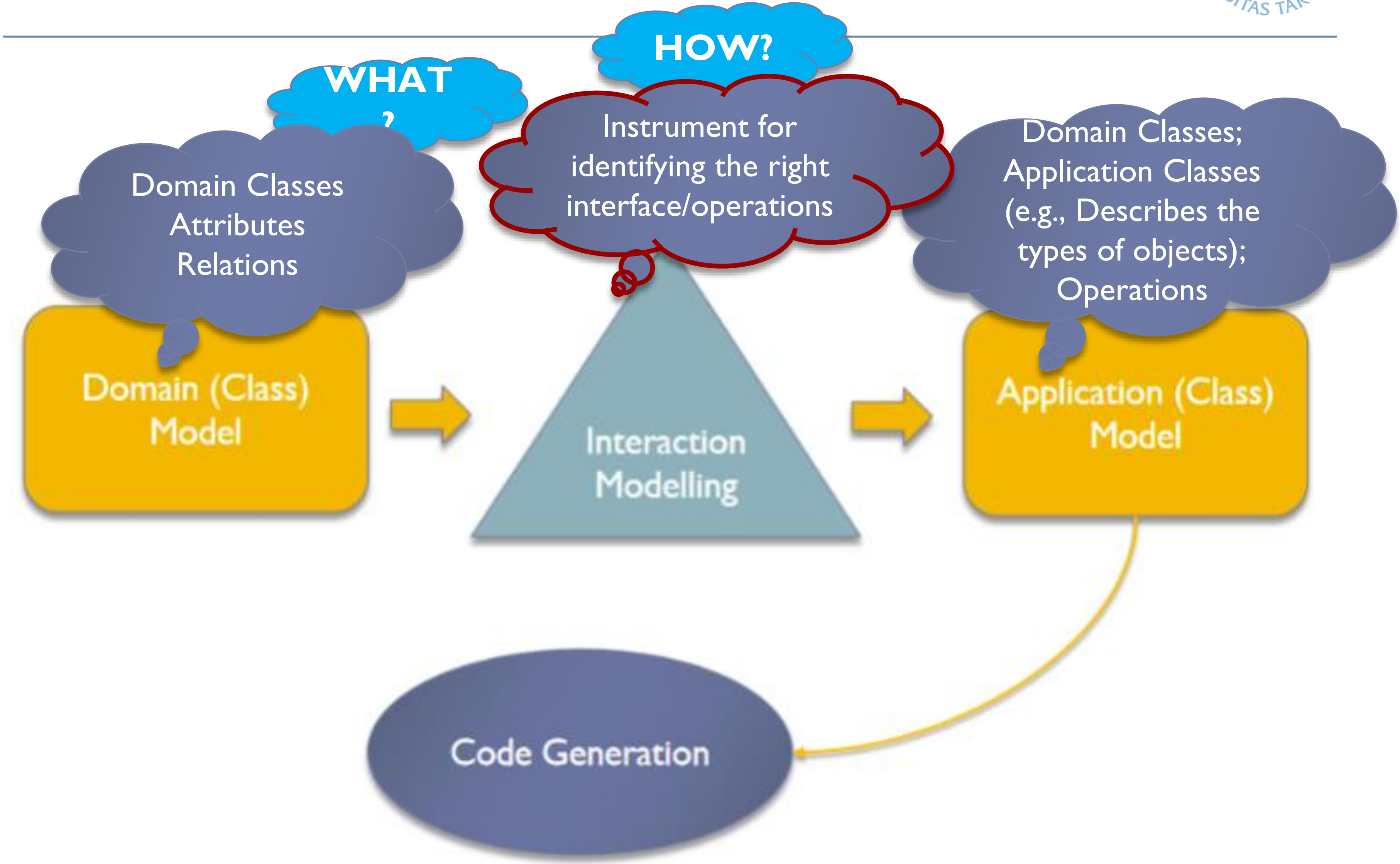
Anastasija Nikiforova

Institute of Computer Science

Based on materials developed by Marlon Dumas, Fabrizio Maria Maggi, Luciano García-Bañuelos

Software Development Methodology




WHAT
?
Domain Classes
Attributes
Relations
HOW?
Instrument for identifying the right interface/operations
Domain Classes; Application Classes (e.g., Describes the types of objects); Operations
Domain (Class) Model
Interaction Modelling
Application (Class) Model
Code Generation

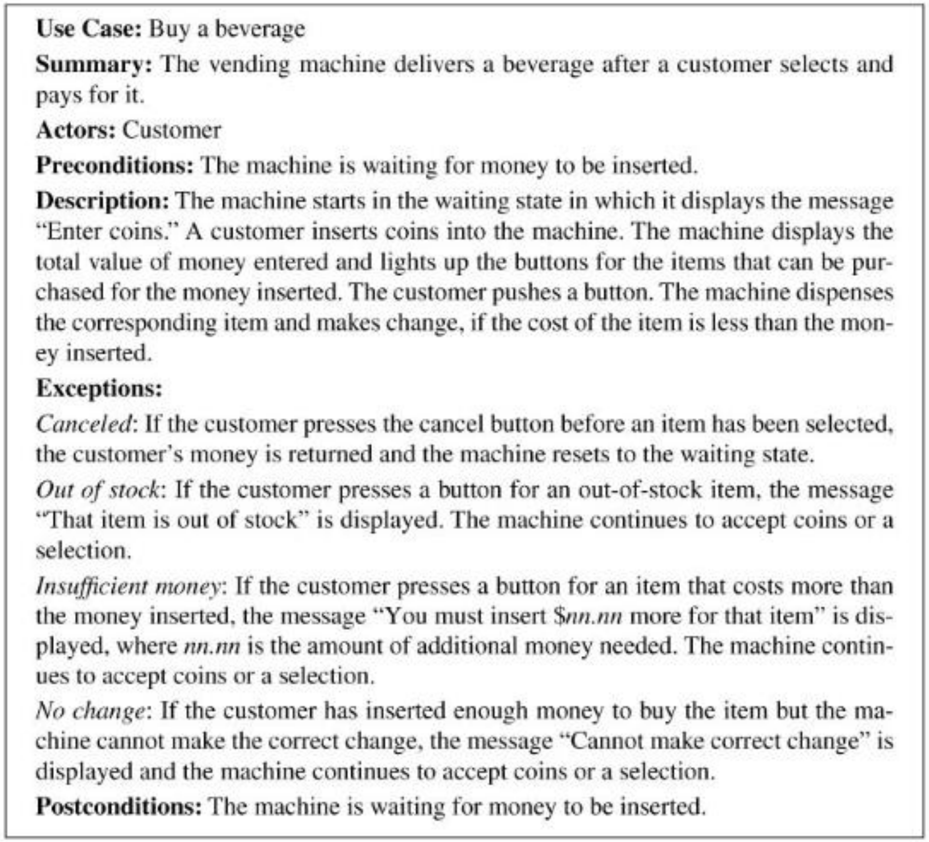
Interaction modelling: overview

HOW?

Interaction modelling

WHAT SERVES AS AN INPUT?

Interaction modelling: Detailing Use Cases with Scenarios



Use Case: Buy a beverage

Summary: The vending machine delivers a beverage after a customer selects and pays for it.

Actors: Customer

Preconditions: The machine is waiting for money to be inserted.

Description: The machine starts in the waiting state in which it displays the message "Enter coins." A customer inserts coins into the machine. The machine displays the total value of money entered and lights up the buttons for the items that can be purchased for the money inserted. The customer pushes a button. The machine dispenses the corresponding item and makes change, if the cost of the item is less than the money inserted.

Exceptions:

Canceled: If the customer presses the cancel button before an item has been selected, the customer's money is returned and the machine resets to the waiting state.

Out of stock: If the customer presses a button for an out-of-stock item, the message "That item is out of stock" is displayed. The machine continues to accept coins or a selection.

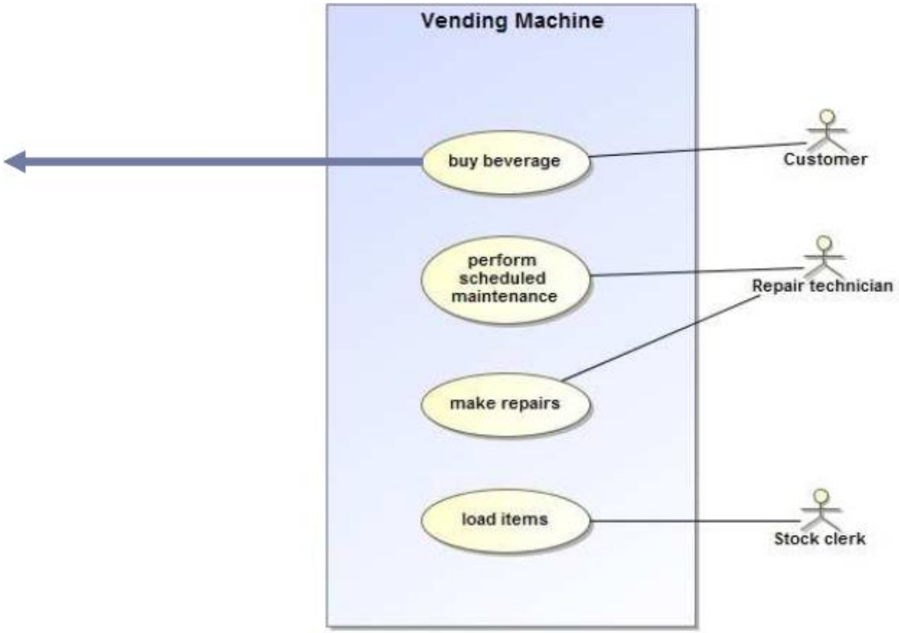
Insufficient money: If the customer presses a button for an item that costs more than the money inserted, the message "You must insert $*nn.nn* more for that item" is displayed, where *nn.nn* is the amount of additional money needed. The machine continues to accept coins or a selection.

No change: If the customer has inserted enough money to buy the item but the machine cannot make the correct change, the message "Cannot make correct change" is displayed and the machine continues to accept coins or a selection.

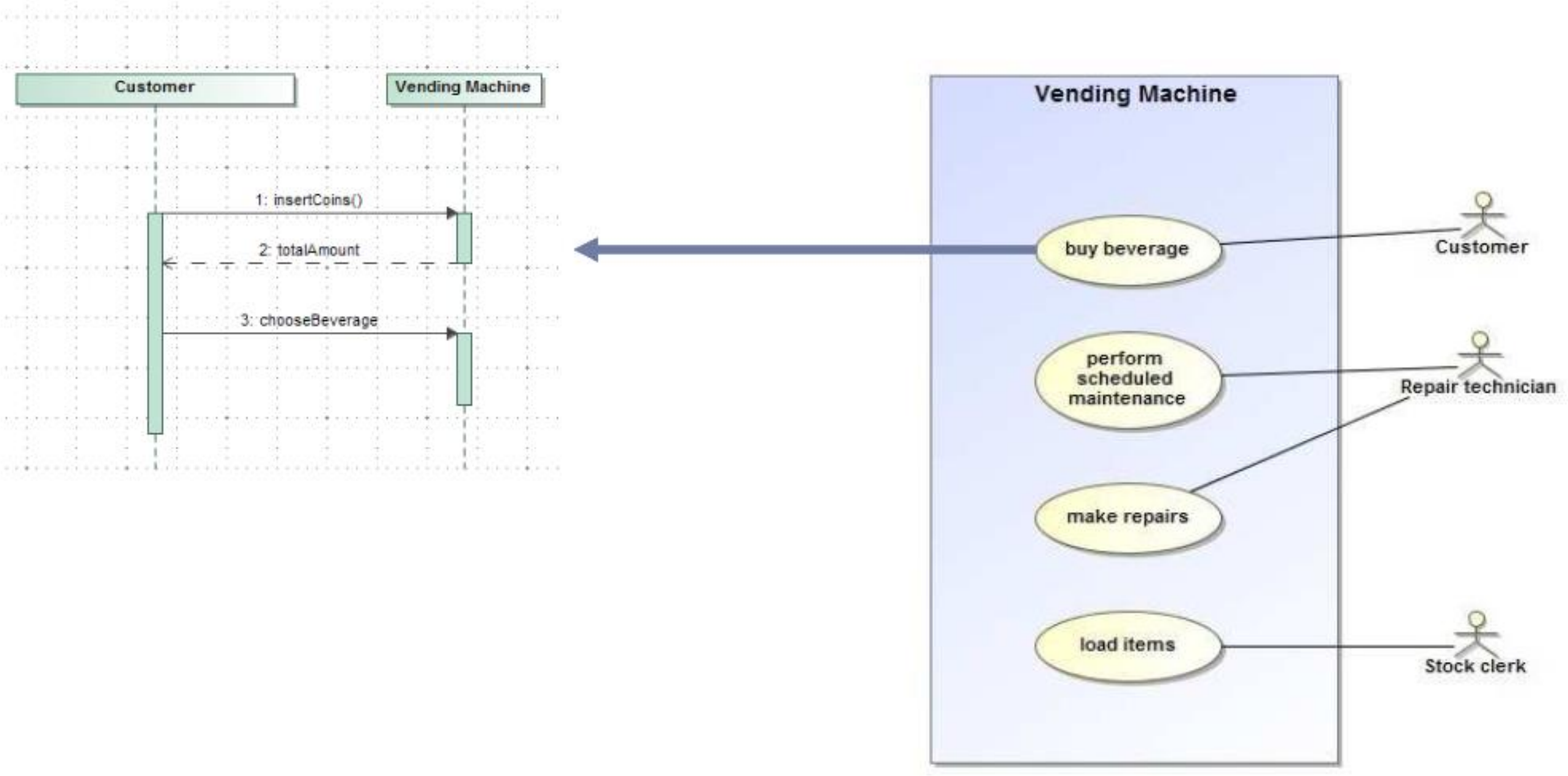
Postconditions: The machine is waiting for money to be inserted.

Figure 7.2 Use case description. A use case brings together all of the behavior relevant to a slice of system functionality.

Object-Oriented Modeling and Design with UML, Second Edition by Michael Blaha and James Rumbaugh. ISBN 0-13-1-015920-4. © 2005 Pearson Education, Inc., Upper Saddle River, NJ. All rights reserved.



Interaction Modelling



Sequence Diagrams as a support to define and document interfaces

!!! The creation of a sequence diagram should be driven by the objective of writing interfaces for the classes of a domain model by reasoning on their interactions in the implementation of the required functionalities

Sequence Diagrams as a support to define and document interfaces

- ➢ A sequence diagram **is used to support the definition of the interfaces** through the **identification of the operations that the classes need to expose for implementing the required functionalities**

- ➢ A sequence diagram **is used to document where the identified operations come into play in the implementation of the required functionalities**

Sequence Diagrams at a glance

Dimensions: object and time

Object Dimension

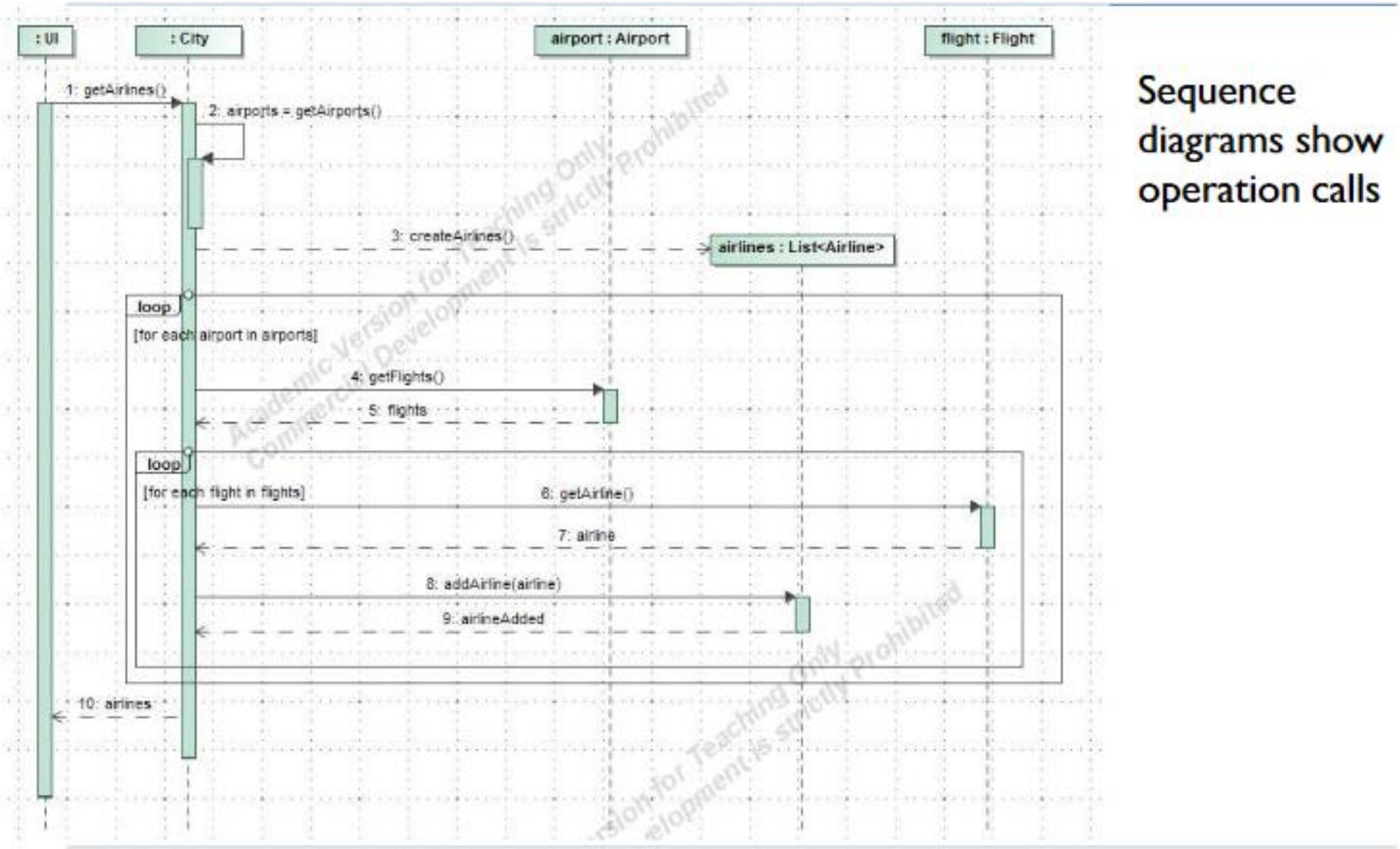
- The horizontal axis shows the elements that are involved in the interaction
- Conventionally, the objects involved in the operation are listed from left to right according to when they take part in the message sequence.

Time Dimension

- The vertical axis represents time proceedings (or progressing) down the page.
- Time in a sequence diagram is all a **about <u>ordering</u>, not duration** ⇒ the vertical space in an interaction diagram is not relevant for the duration of the interaction.

Source: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-sequence-diagram/

Sequence Diagrams



Sequence diagrams show operation calls

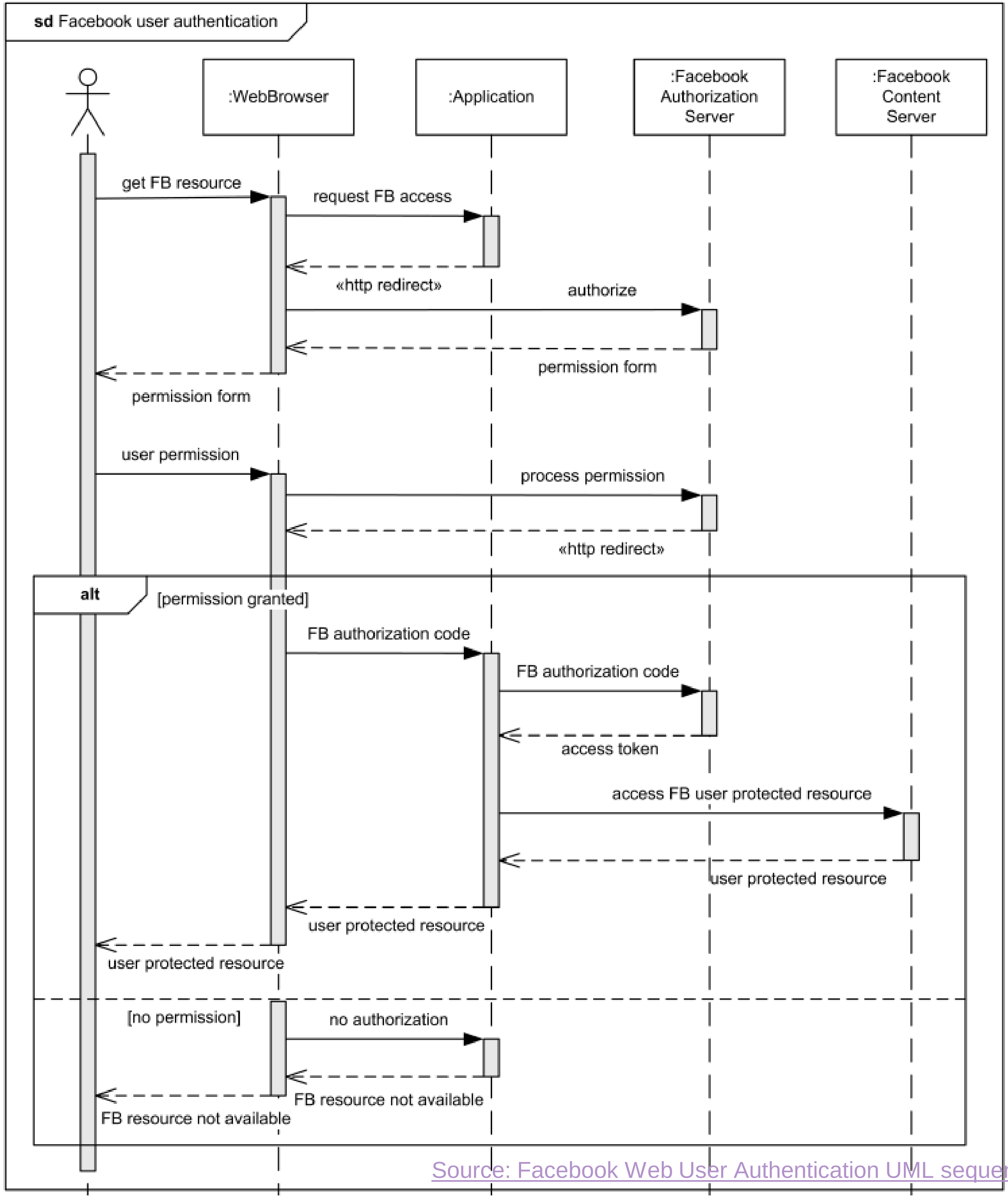
Example

- **Facebook (FB) user can be authenticated in a web application to allow access to his/her FB resources.**

- **Facebook uses OAuth 2.0 protocol framework which enables web application (called "client"), which is usually not the FB resource owner but is acting on the FB user's behalf, to request access to resources controlled by the FB user and hosted by the FB server. Instead of using the FB user credentials to access protected resources, the web application obtains an access token.**
- **Web application should be registered by Facebook to have an application ID (client_id) and secret (client_secret).**
- **When request to some protected Facebook resources is received, web browser ("user agent") is redirected to Facebook's authorization server with application ID and the URL the user should be redirected back to after the authorization process.**
- **User receives back Request for Permission form.**
- **If the user authorizes the application to get his/her data, Facebook authorization server redirects back to the URI that was specified before together with authorization code ("verification string").**
- **The authorization code can be exchanged by web application for an OAuth access token.**
- **If web application obtains the access token for a FB user, it can perform authorized requests on behalf of that FB user by including the access token in the Facebook Graph API requests.**
- **If the user did not authorize web application, Facebook issues redirect request to the URI specified before, and adds the error_reason parameter to notify the web application that authorization request was denied.**

Source: Facebook Web User Authentication UML sequence diagram example. (uml-diagrams.org)

Example



Source: Facebook Web User Authentication UML sequence diagram example. (uml-diagrams.org)

Relating Interaction Diagrams

What if I want to relate or link diagrams? ⇒ an interaction occurrence!

An interaction occurrence (also "interaction use") is a reference to an interaction within another interaction.

When to use?

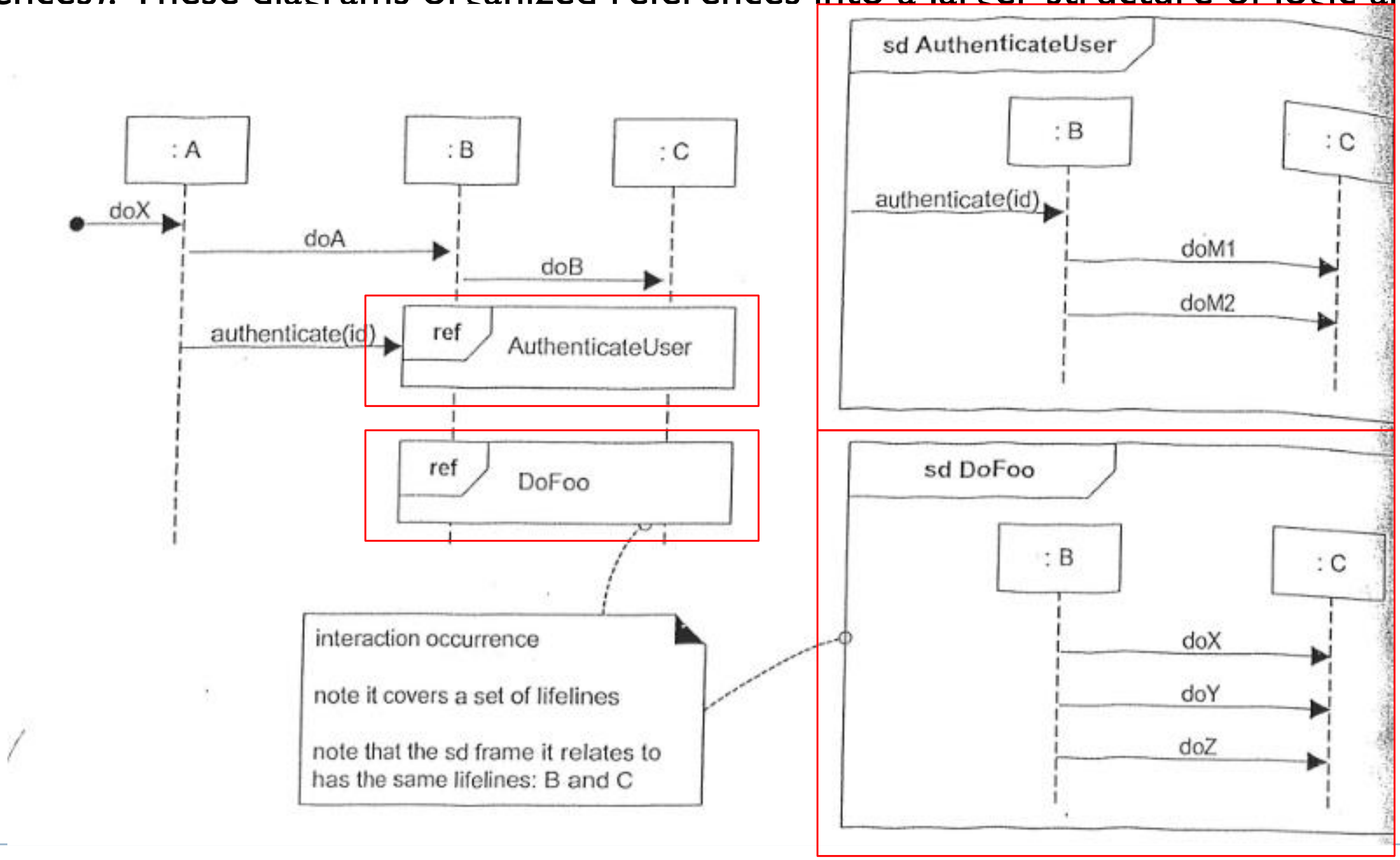
When you want to simplify a diagram and factor out a portion into another diagram, or there is a reusable interaction occurrence.

Created with **two related frames**:

- ○ a frame around an entire sequence diagram, labeled with the tag ***sd*** and a name, such as AuthenticateUser
- ○ a frame tagged ***ref***, called a ***reference***, that refers to another named sequence diagram; it is the actual interaction occurrence

Relating Interaction Diagrams

Interaction overview diagrams also contain a set of reference frames (interaction occurrences). These diagrams organized references into a larger structure of logic and process flow.



The Entity-Control-Boundary Pattern

The Entity-Control-Boundary (ECB) aka the entity-control-boundary (EBC) is a ECB is a simplification of the Model-View-Controller Pattern.

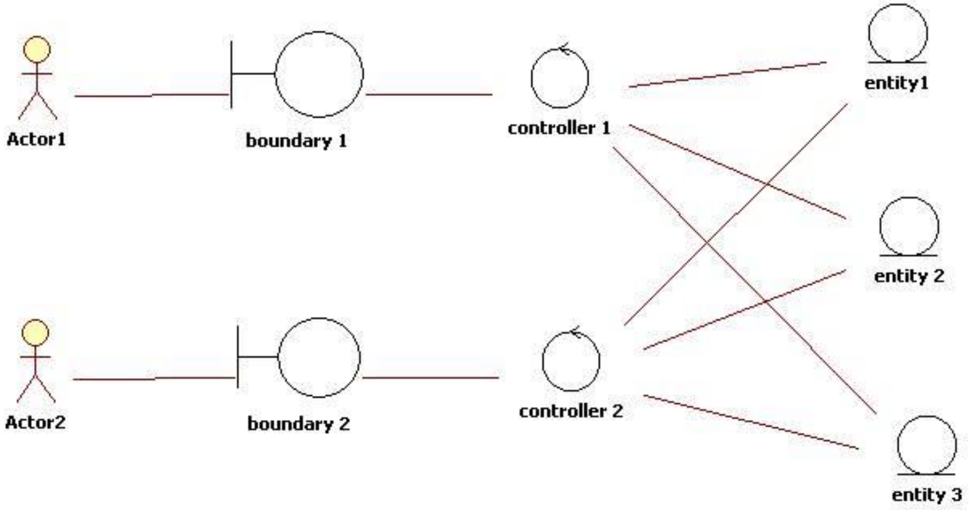
ECB partitions the system into three types of classes

what are they?

The Entity-Control-Boundary Pattern

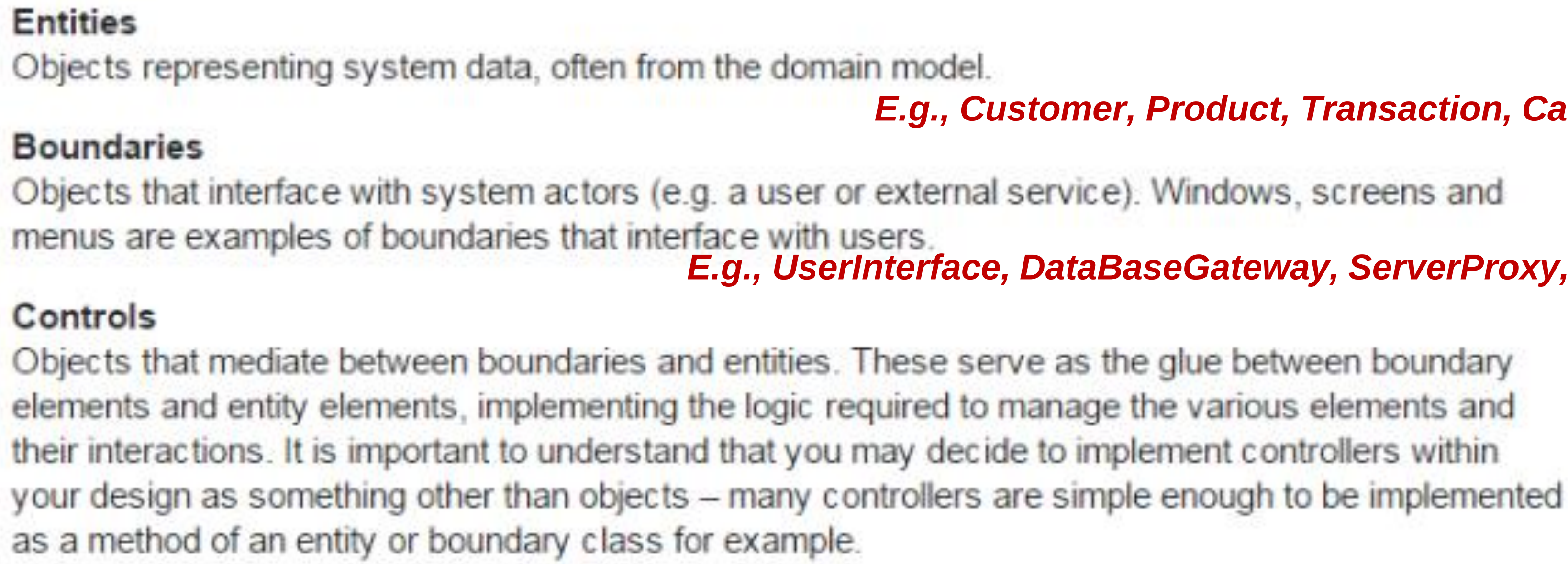
The Entity-Control-Boundary (ECB) aka the entity-control-boundary (EBC) is a ECB is a simplification of the Model-View-Controller Pattern.

ECB partitions the system into three types of classes – entity, control, and boundary are official UML class stereotypes.



Source: The Entity-Control-Boundary Pattern (sjsu.edu)

The Entity-Control-Boundary Pattern



Entities

Objects representing system data, often from the domain model.

E.g., Customer, Product, Transaction, Cart, etc.

Boundaries

Objects that interface with system actors (e.g. a user or external service). Windows, screens and menus are examples of boundaries that interface with users.

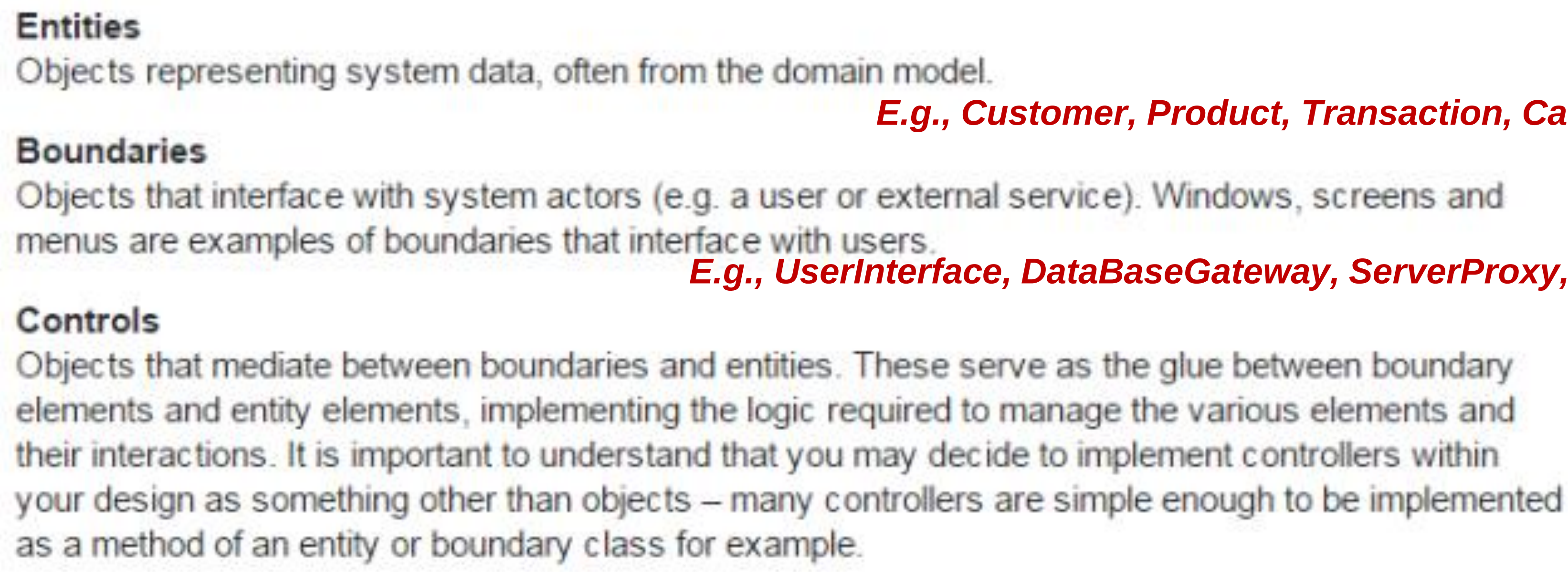
E.g., UserInterface, DataBaseGateway, ServerProxy, etc.

Controls

Objects that mediate between boundaries and entities. These serve as the glue between boundary elements and entity elements, implementing the logic required to manage the various elements and their interactions. It is important to understand that you may decide to implement controllers within your design as something other than objects – many controllers are simple enough to be implemented as a method of an entity or boundary class for example.

They orchestrate the execution of commands coming from the boundary by interacting with entity and boundary objects. ***Controls often correspond to use cases*** *and map to* <u>*use case controllers*</u> *in the design model.*

The Entity-Control-Boundary Pattern



Entities

Objects representing system data, often from the domain model.

E.g., Customer, Product, Transaction, Cart, etc.

Boundaries

Objects that interface with system actors (e.g. a user or external service). Windows, screens and menus are examples of boundaries that interface with users.

E.g., UserInterface, DataBaseGateway, ServerProxy, etc.

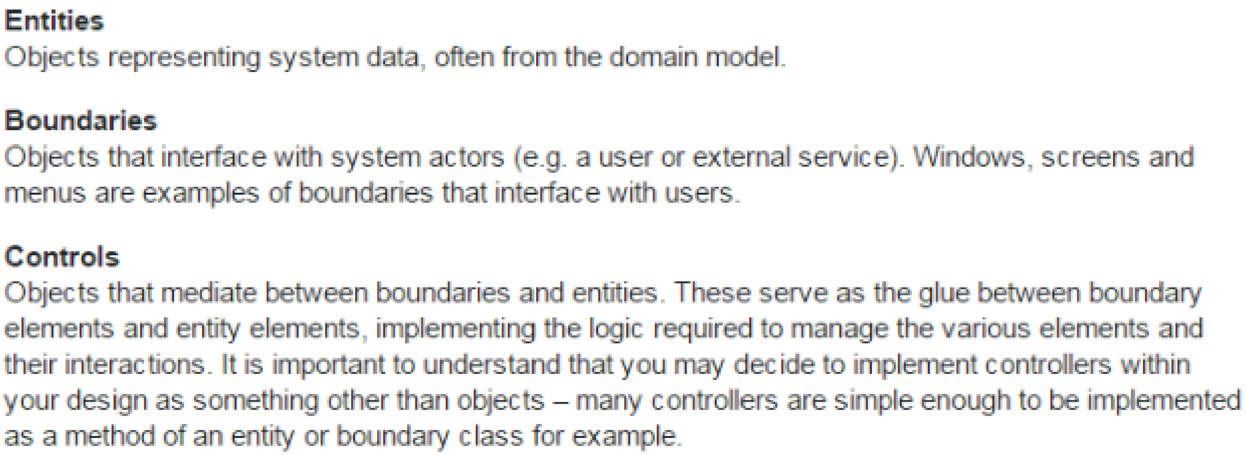
Controls

Objects that mediate between boundaries and entities. These serve as the glue between boundary elements and entity elements, implementing the logic required to manage the various elements and their interactions. It is important to understand that you may decide to implement controllers within your design as something other than objects – many controllers are simple enough to be implemented as a method of an entity or boundary class for example.

They orchestrate the execution of commands coming from the boundary by interacting with entity and boundary objects. ***Controls often correspond to use cases*** *and map to* *<u>use case controllers</u>* *in the design model.*

******Associated with every control in the analysis model is a statechart diagram representing the control's internal logic.***

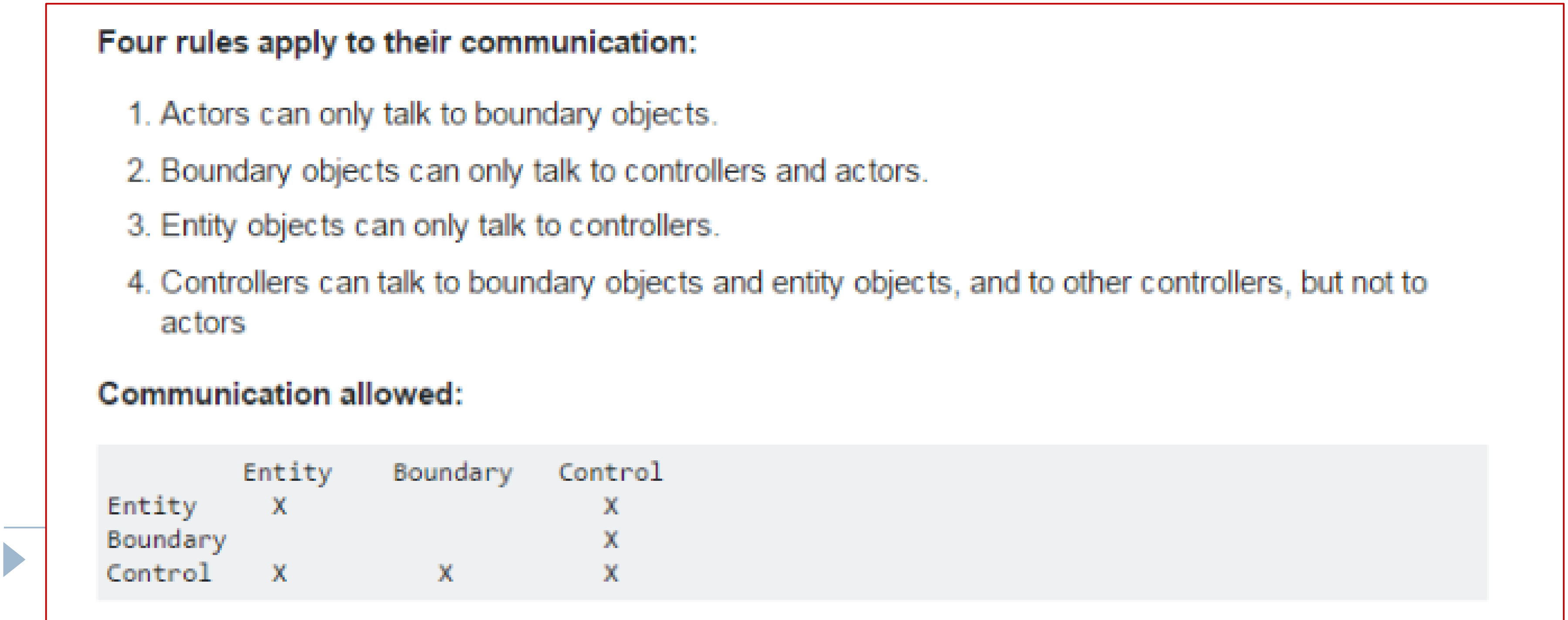
The Entity-Control-Boundary Pattern



Entities
Objects representing system data, often from the domain model.

Boundaries
Objects that interface with system actors (e.g. a user or external service). Windows, screens and menus are examples of boundaries that interface with users.

Controls
Objects that mediate between boundaries and entities. These serve as the glue between boundary elements and entity elements, implementing the logic required to manage the various elements and their interactions. It is important to understand that you may decide to implement controllers within your design as something other than objects – many controllers are simple enough to be implemented as a method of an entity or boundary class for example.



Four rules apply to their communication:

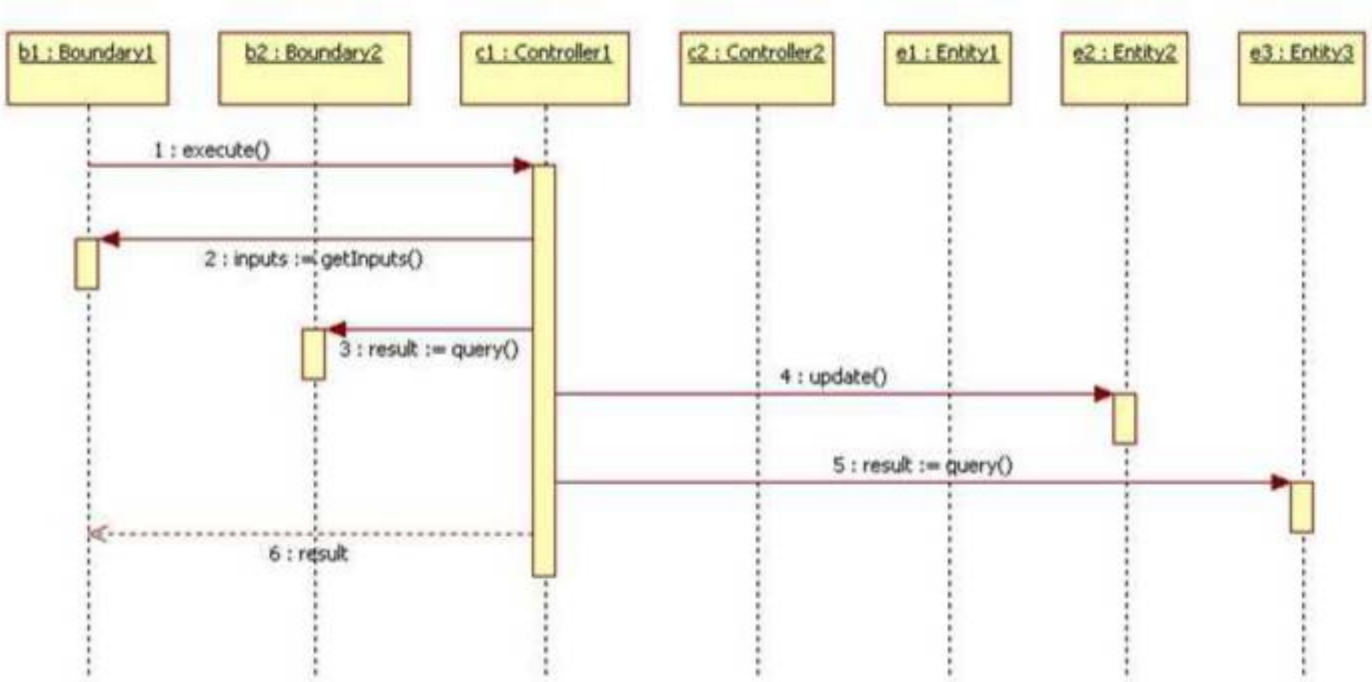
1. Actors can only talk to boundary objects.
2. Boundary objects can only talk to controllers and actors.
3. Entity objects can only talk to controllers.
4. Controllers can talk to boundary objects and entity objects, and to other controllers, but not to actors

Communication allowed:

	Entity	Boundary	Control
Entity	X		X
Boundary			X
Control	X	X	X

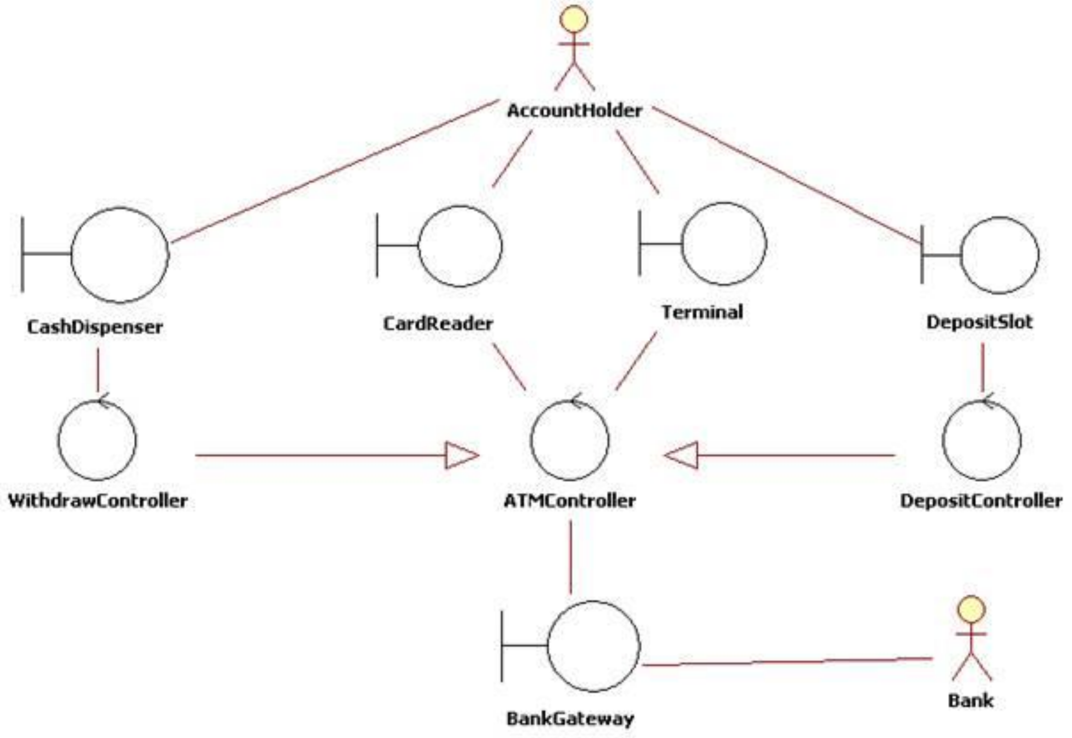
The Entity-Control-Boundary Pattern

- **Actors interact with boundary objects.**
- **Boundary objects issue commands to controller objects.**
- **Controller objects may send queries back to the boundary objects to get more information from the actors.**
- **Controllers then update entities.**



Example: ATM

The Simple ATM allows account holders to make deposits to and withdraw funds from any accounts held at any branch of the Bank of Antarctica.



Analysis model

Source: Analysis Model for a Simple Automatic Teller Machine (ATM) (sjsu.edu)

Example: ATM

The Simple ATM allows account holders to make deposits to and withdraw funds from any accounts held at any branch of the Bank of Antarctica.

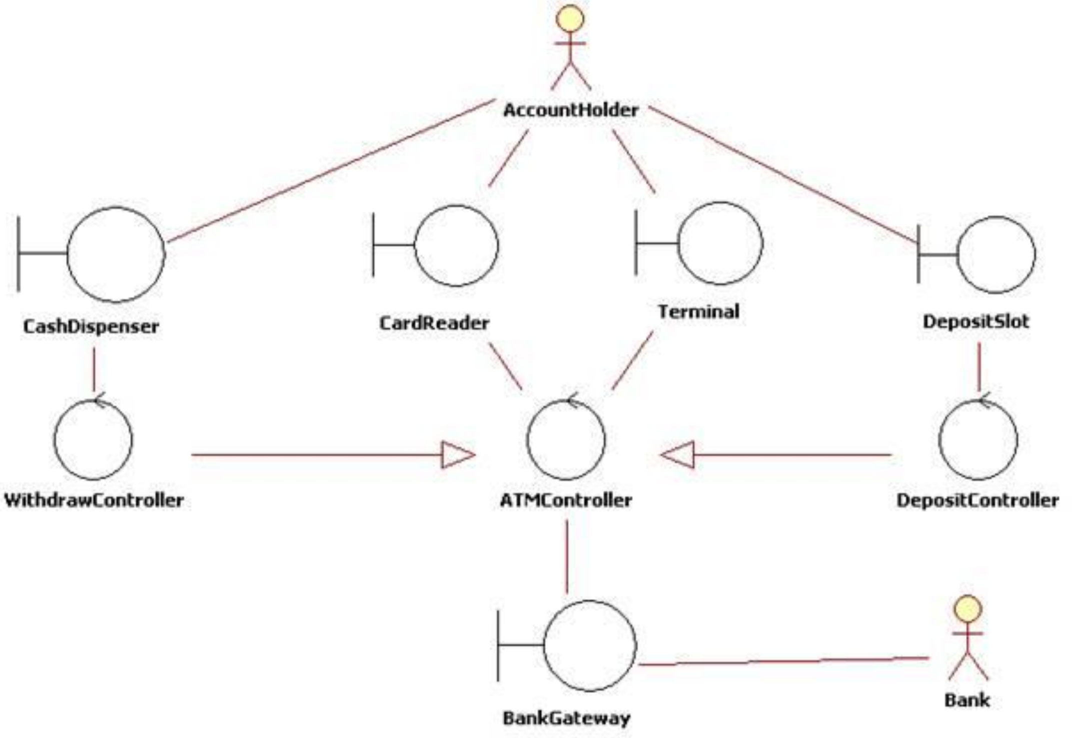
Boundaries

interfaces:
Cash Dispenser, Card Reader, Terminal, DepositSlot

Controllers

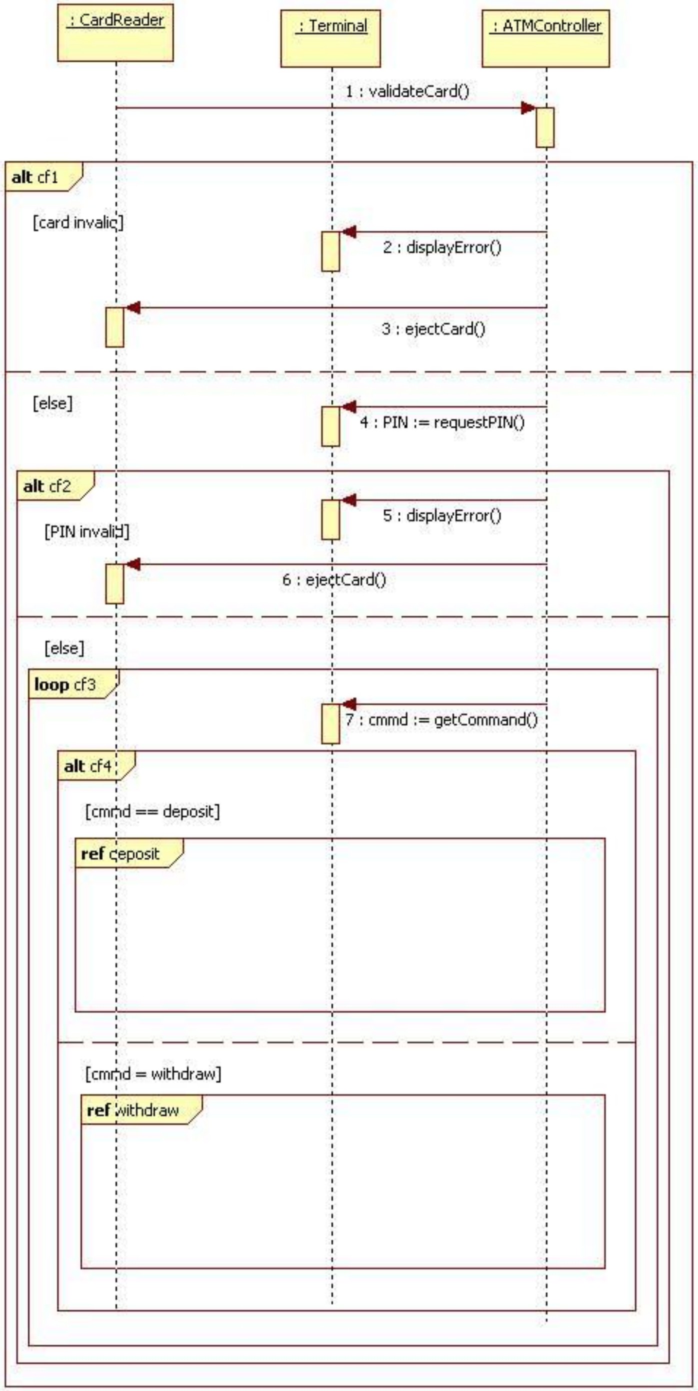
ATM controller, WithdrawController, DepositController

Sequence diagrams

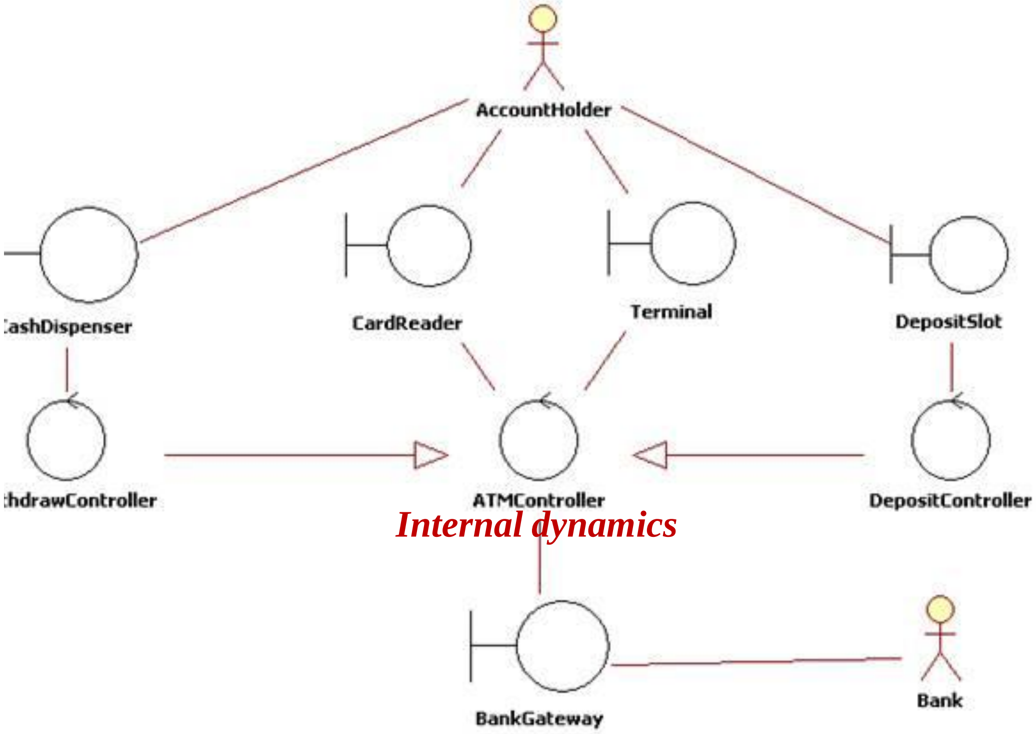


Analysis model

Source: Analysis Model for a Simple Automatic Teller Machine (ATM) (sjsu.edu)



make deposits to and withdraw funds from any accounts held at any



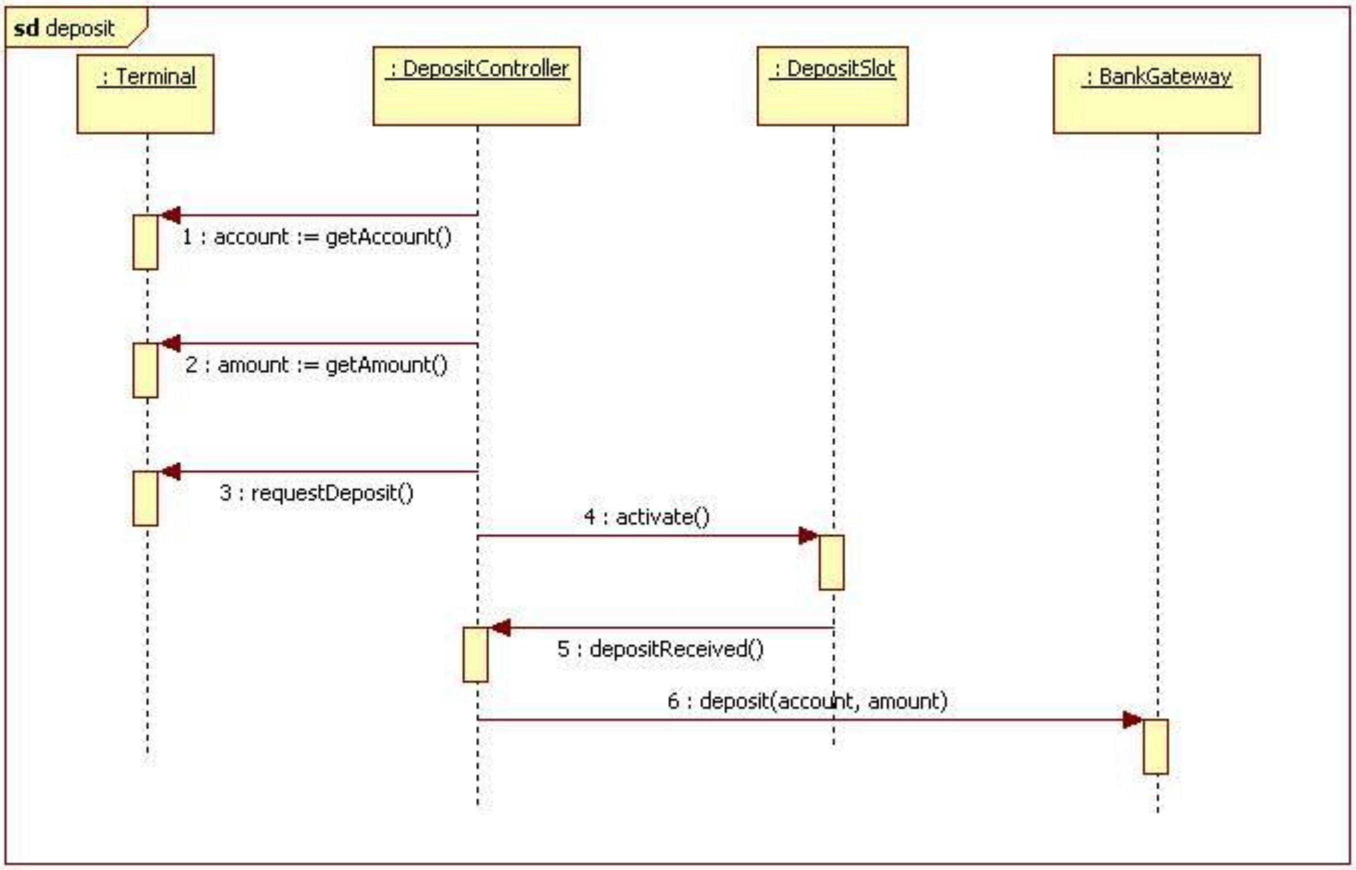
Internal dynamics

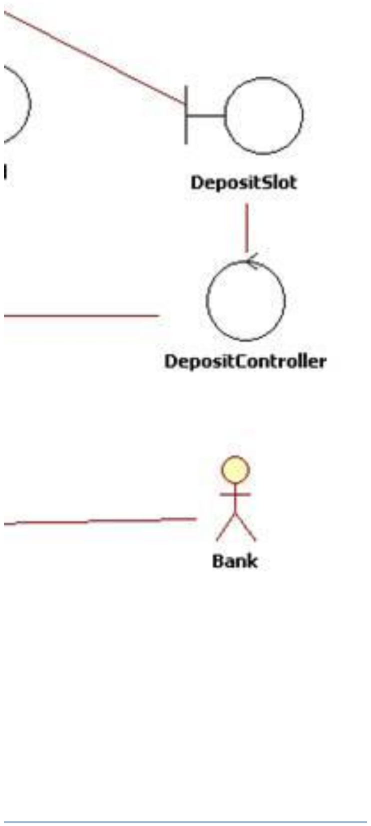
Analysis model

Source: Analysis Model for a Simple Automatic Teller Machine (ATM) (sjsu.edu)

Example: ATM

The Simple ATM allows account holders to make deposits to and withdraw funds from any accounts held at any branch of the Bank of Antarctica.





[Source: Analysis Model for a Simple Automatic Teller Machine (ATM) (sjsu.edu)](https://)

Example: ATM

The Simple ATM allows account holders to make deposits to and withdraw funds from any accounts held at any branch of the Bank of Antarctica.

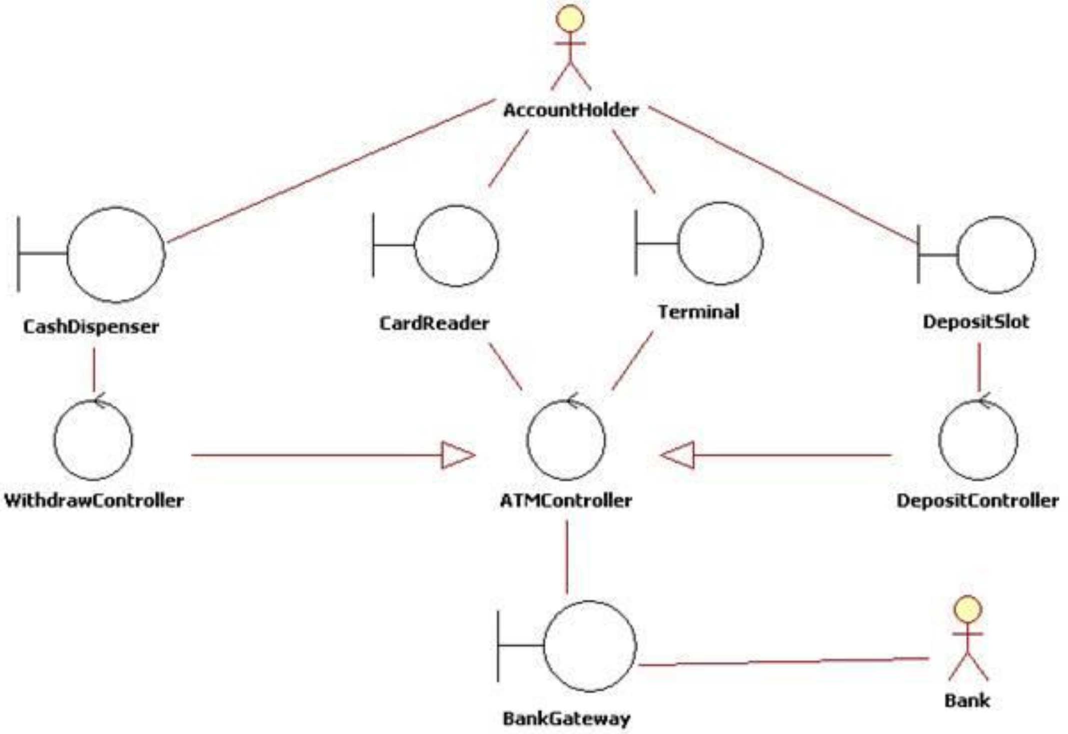
Boundaries

interfaces:
Cash Dispenser, Card Reader, Terminal, DepositSlot

Controllers

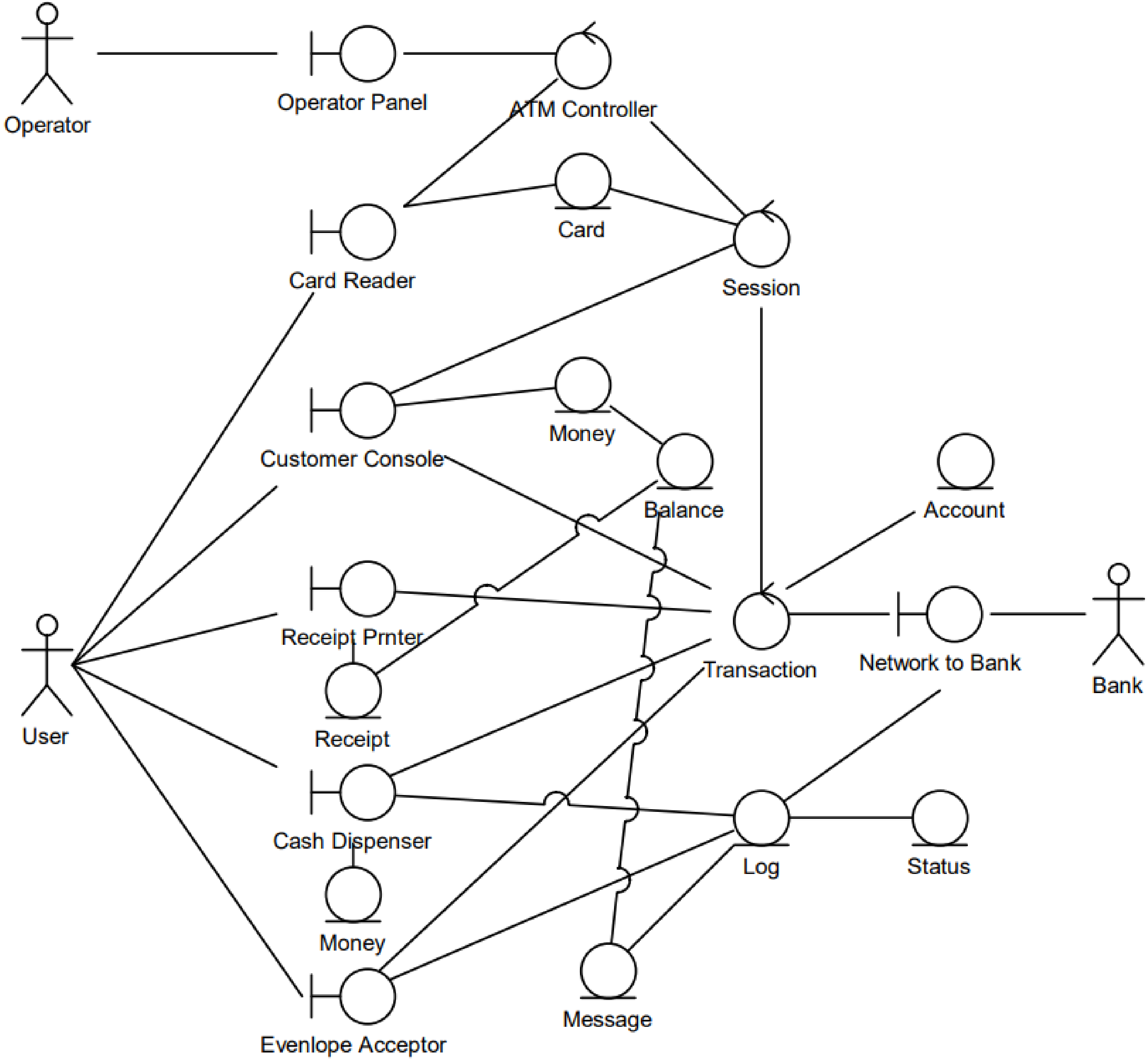
ATM controller, WithdrawController, DepositController

Sequence diagrams

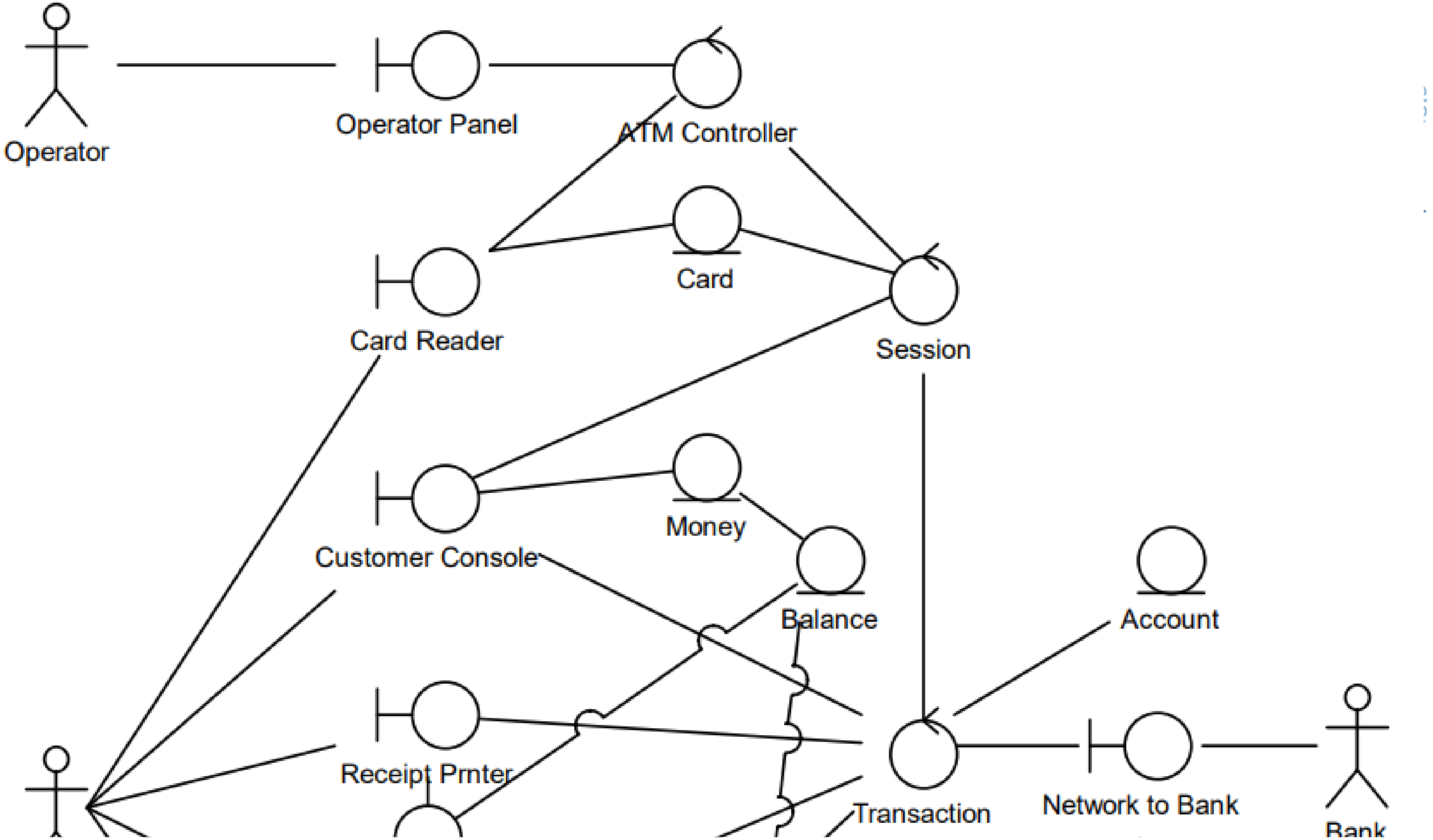


Will we create Sequence Diagram for WithdrawController?

Source: Analysis Model for a Simple Automatic Teller Machine (ATM) (sjsu.edu)

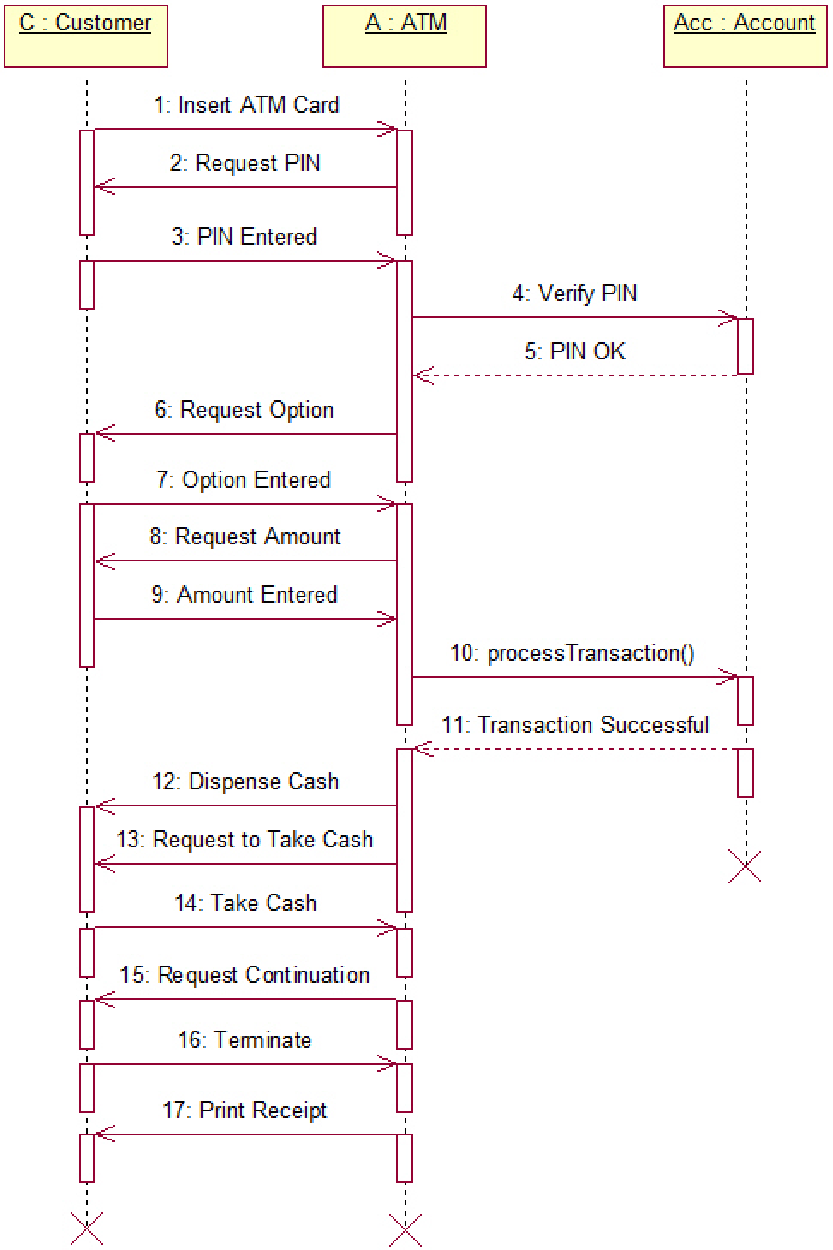


Cource: Microsoft Word - Week9 (csun.edu)



Boundary classes	Controller classes	Entity classes
Class CardReader	Class ATM Controller	Class Account
Class CashDispenser	Class Session	Class Card
Class CustomerConsole	Class Transaction	Class Receipt
Class EnvelopeAcceptor	Class Withdrawal	Class Log
Class NetworkToBank	Class Deposit	Class Money
Class OperatorPanel	Class Transfer	Class Status
Class ReceiptPrinter	Class Inquiry	Class Message

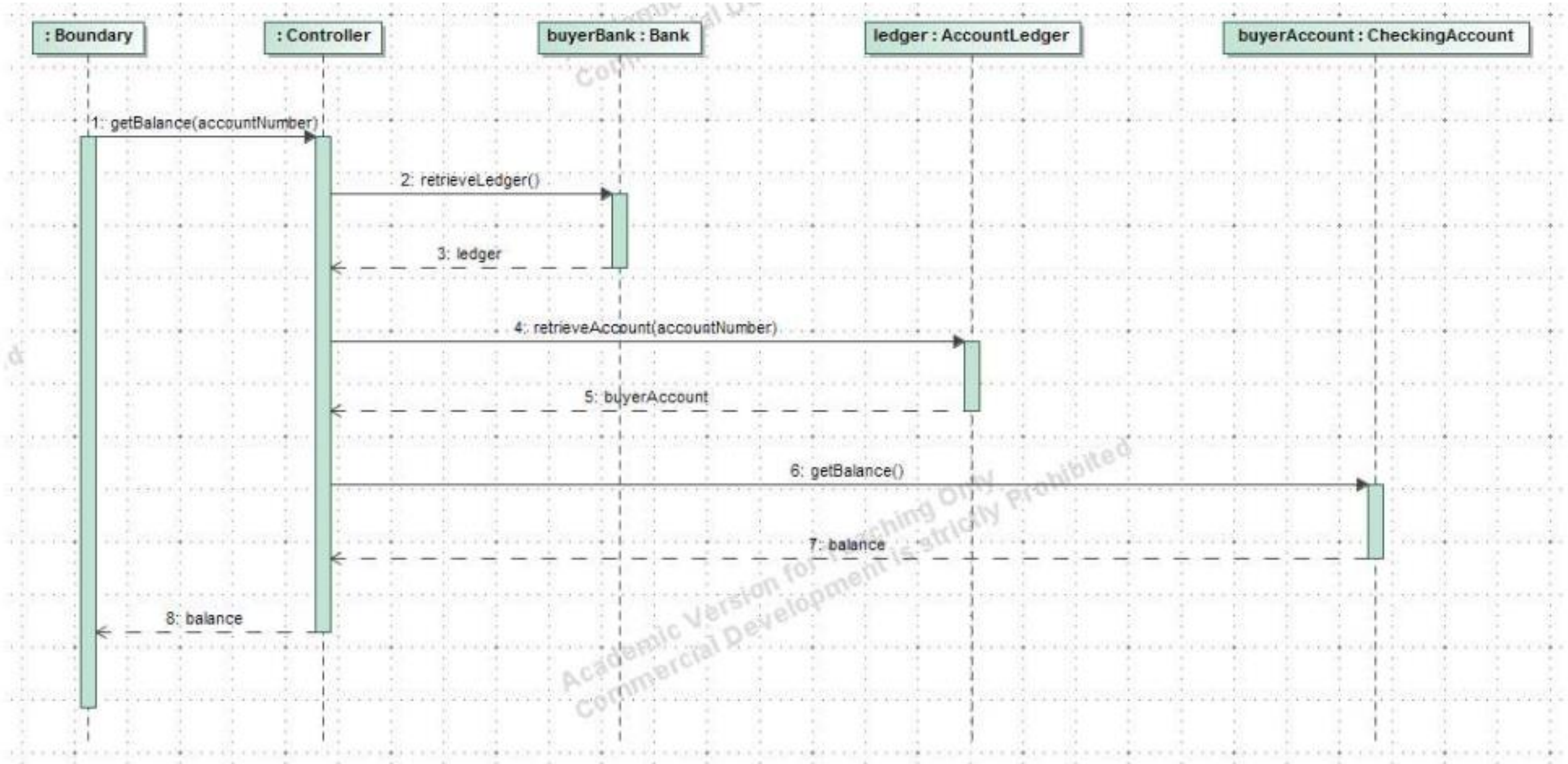
Cource: Microsoft Word - Week9 (csun.edu)



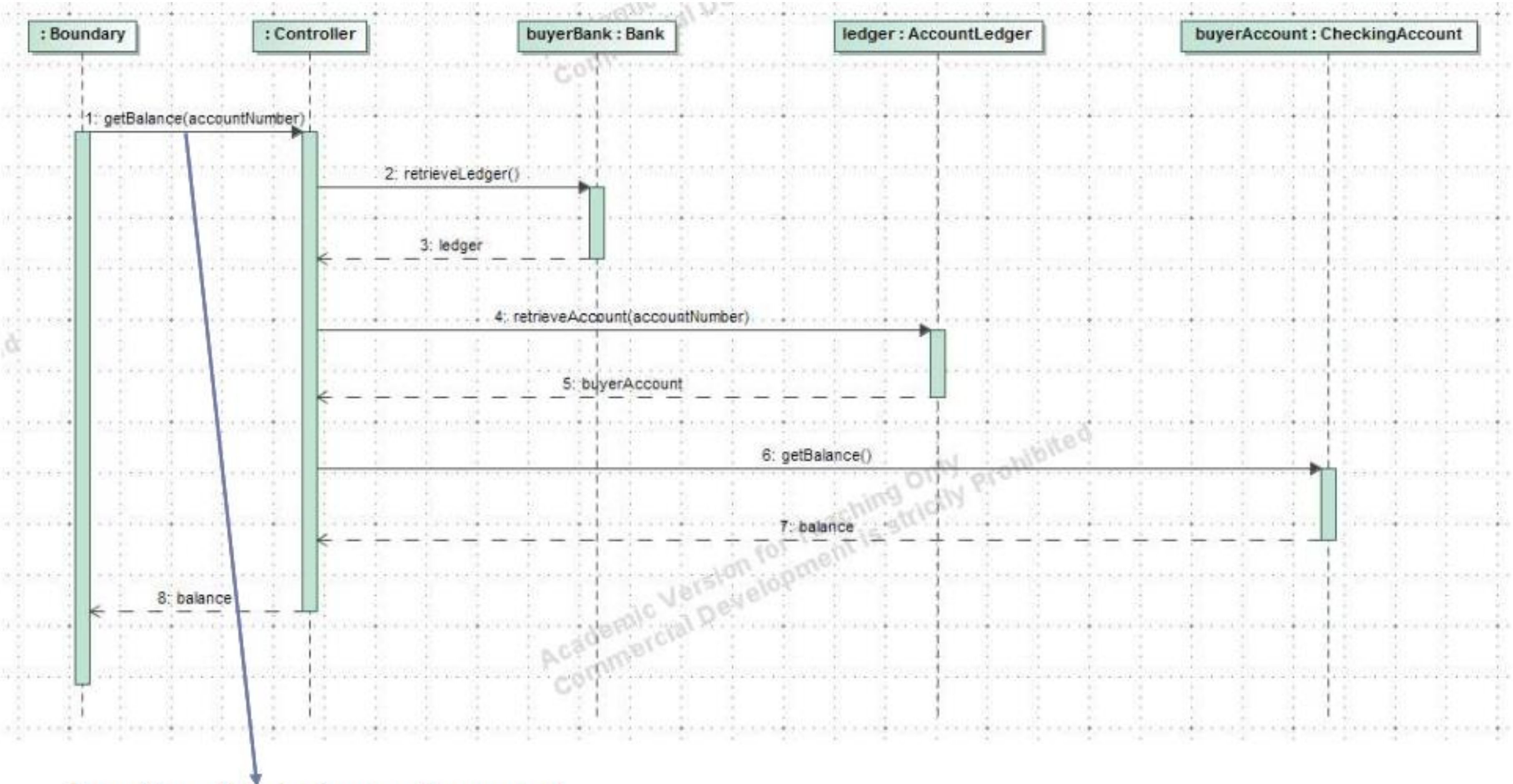
Note, different level of detail and scope do matter!

ATM UML Diagrams (startertutorials.com)

The Entity-Control-Boundary Pattern

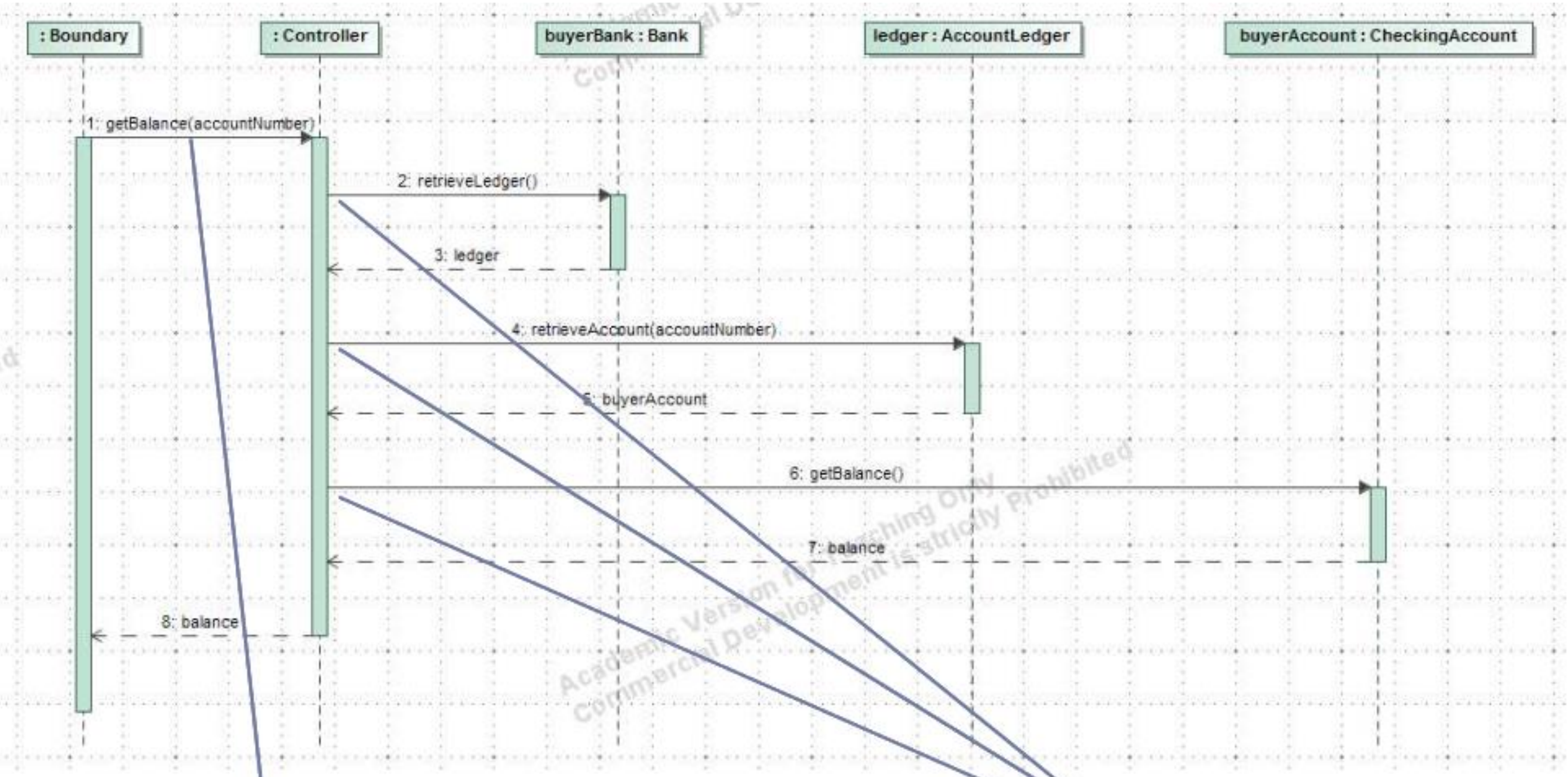


The Entity-Control-Boundary Pattern



Functionality to be implemented

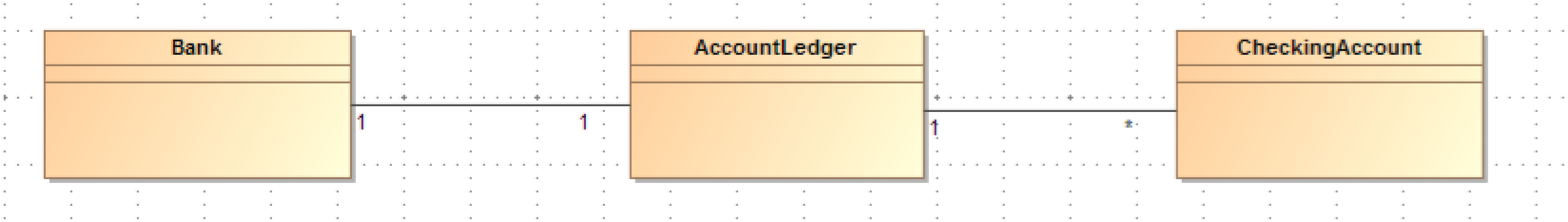
The Entity-Control-Boundary Pattern

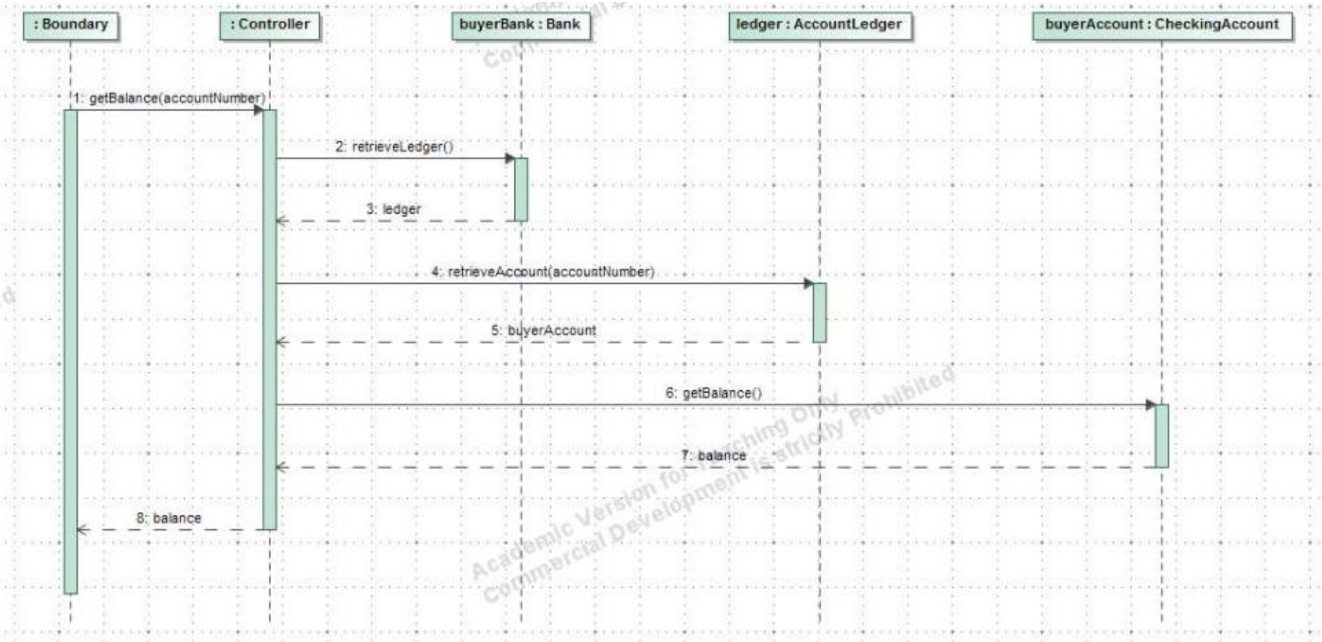


Functionality to be implemented

Most of the calls come from the controller that plays the role of the „orchestrator" in the implementation of the functionality

From a domain to application model





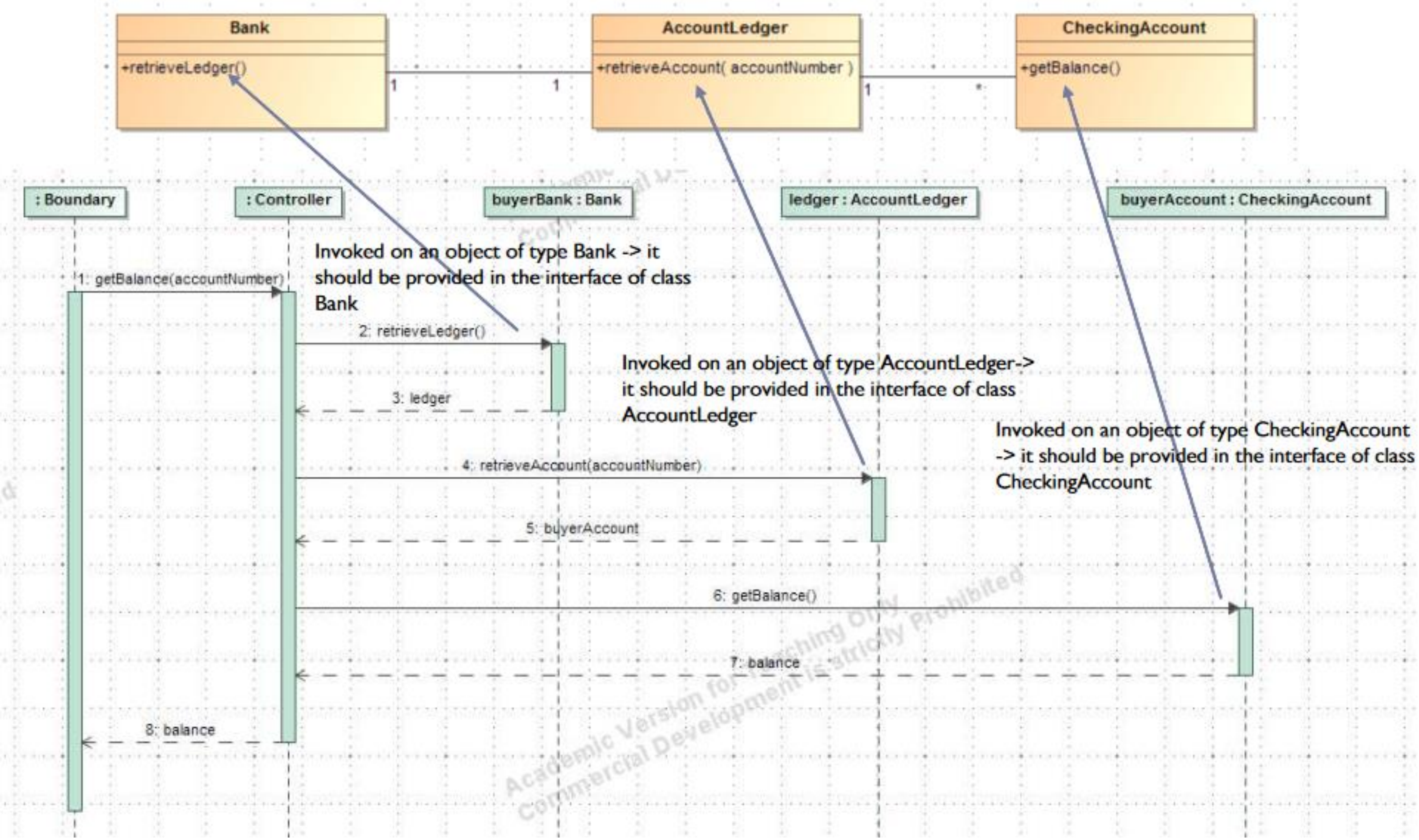
: Boundary
: Controller
buyerBank : Bank
ledger : AccountLedger
buyerAccount : CheckingAccount
1: getBalance(accountNumber)
2: retrieveLedger()
3: ledger
4: retrieveAccount(accountNumber)
5: buyerAccount
6: getBalance()
7: balance
8: balance
Academic Version for Teaching Only
Commercial Development is strictly Prohibited

Application (class) model

Operations/Interfaces

carry
come
drink
go
lead
ask
catch
cook
drive
hit
lift
bake
clap
cry
eat
hop
lock
bite
clean
cut
float
juggle
look
bounce
climb
dance
fly
jump
march
brush
close
dig
fold
kick
mix
build
color
draw
follow
knock
mop
call
comb
dream
give
laugh
open

From a domain to application model

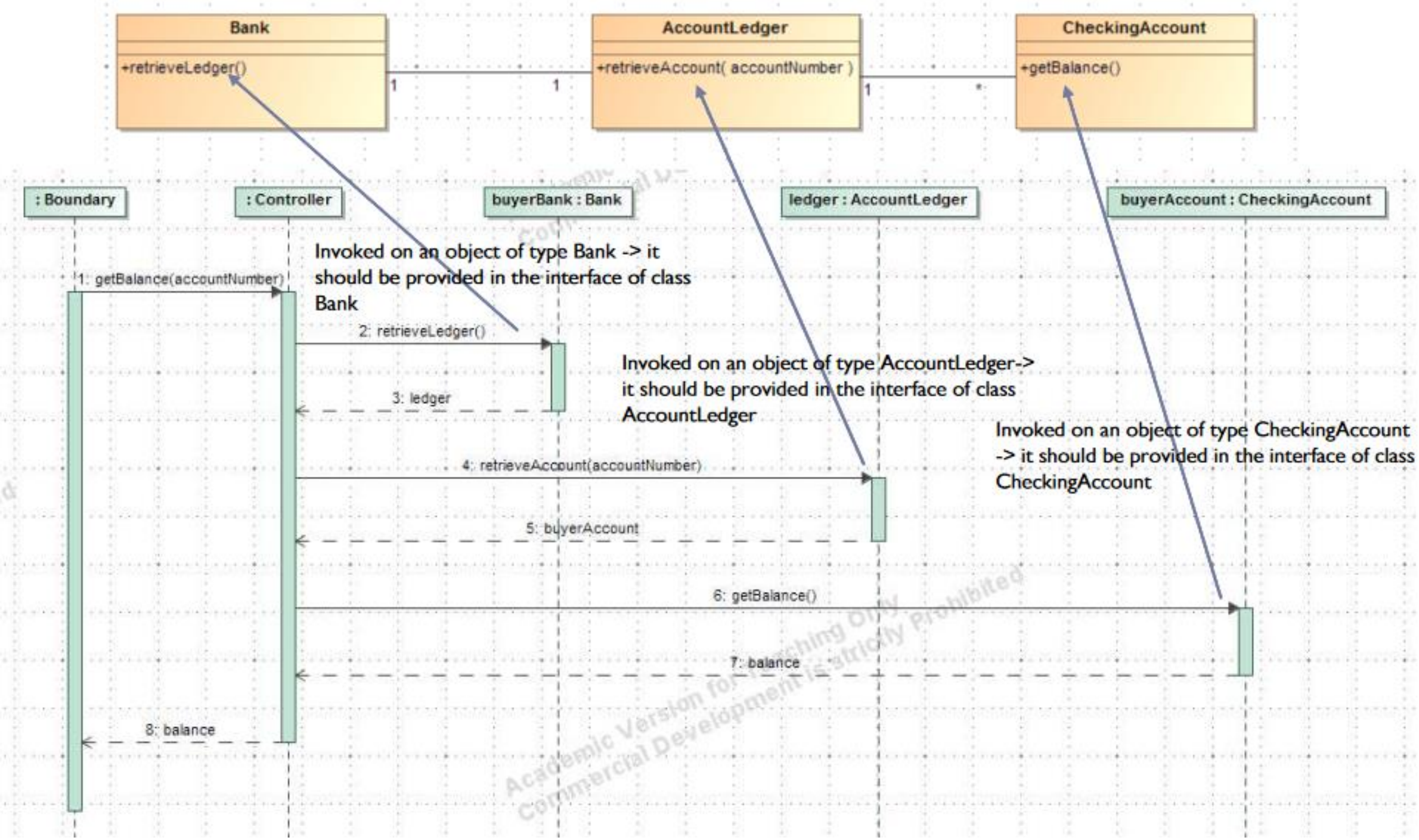


Invoked on an object of type Bank -> it should be provided in the interface of class Bank

Invoked on an object of type AccountLedger-> it should be provided in the interface of class AccountLedger

Invoked on an object of type CheckingAccount -> it should be provided in the interface of class CheckingAccount

From a domain to application model

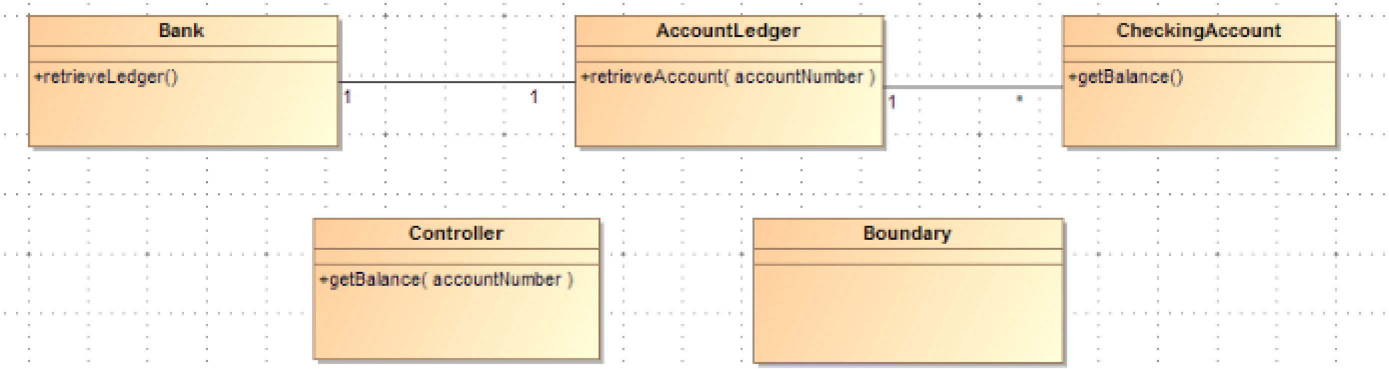


Invoked on an object of type Bank -> it should be provided in the interface of class Bank

Invoked on an object of type AccountLedger-> it should be provided in the interface of class AccountLedger

Invoked on an object of type CheckingAccount -> it should be provided in the interface of class CheckingAccount

From a domain to application model



application model

Application model

- For creating an **application model**, you start from a **domain model** where the **operations** of the classes **are not specified** and you only have **classes representing *entities* of the domain**.
- Then, **through interaction modeling you understand what operations are needed to implement a certain functionality** and what **additional application classes (outside the domain) are needed to implement it** (e.g., Boundary and Controllers).

- **The application model is the class model obtained by adding operations and application classes to the domain model.**

Application model

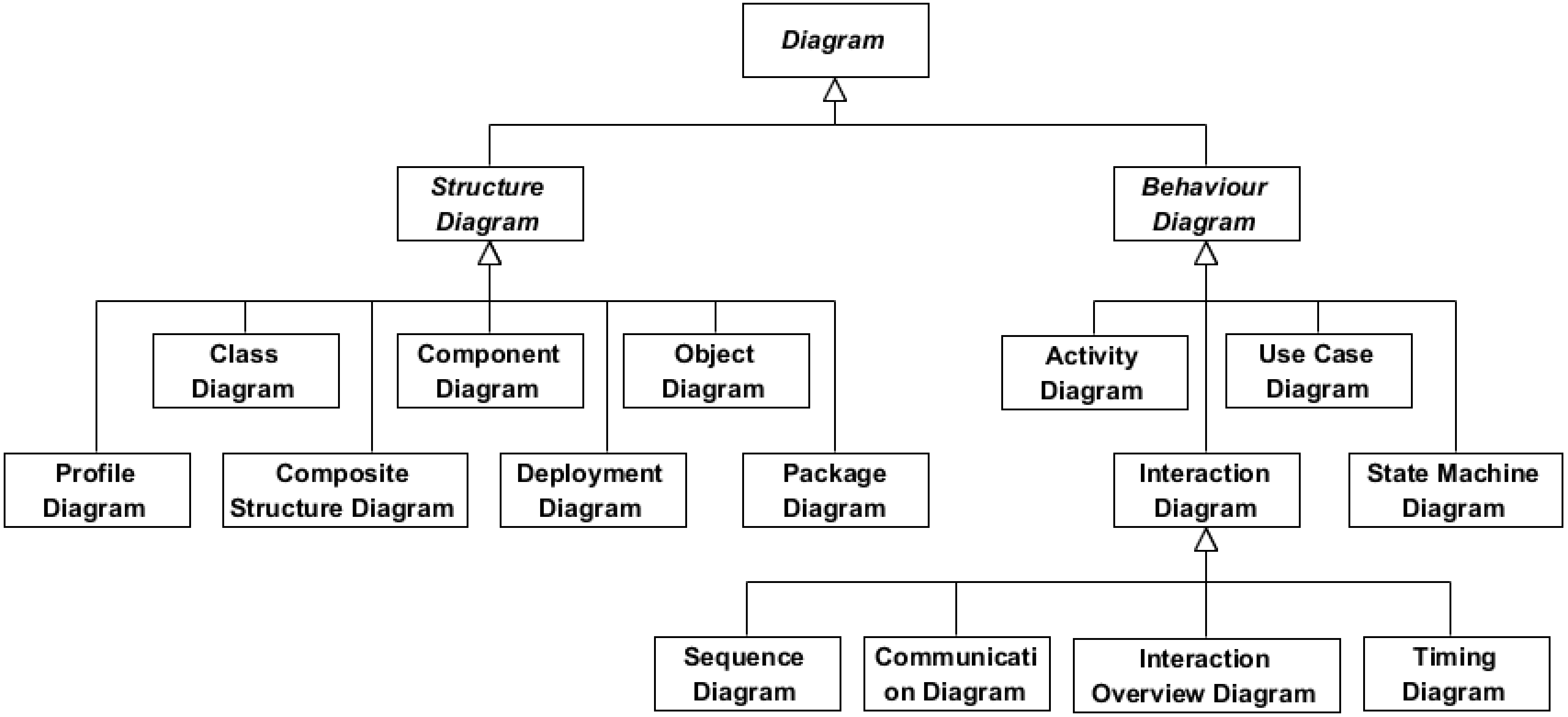
- It is possible to add redundant associations to access the data more efficiently

- It is possible to specify the direction of certain associations

Interaction modelling

- **Interactions** can be modeled **at different levels of abstraction**
  - **At a high level use cases** describe **how a system interacts with outside actors**
    - **Each use case represents a functionality that a system provides to the user**
    - Use cases are **helpful for capturing informal requirements**

- **Sequence diagrams** provide more details about **which operations need to be invoked in a specific scenario**

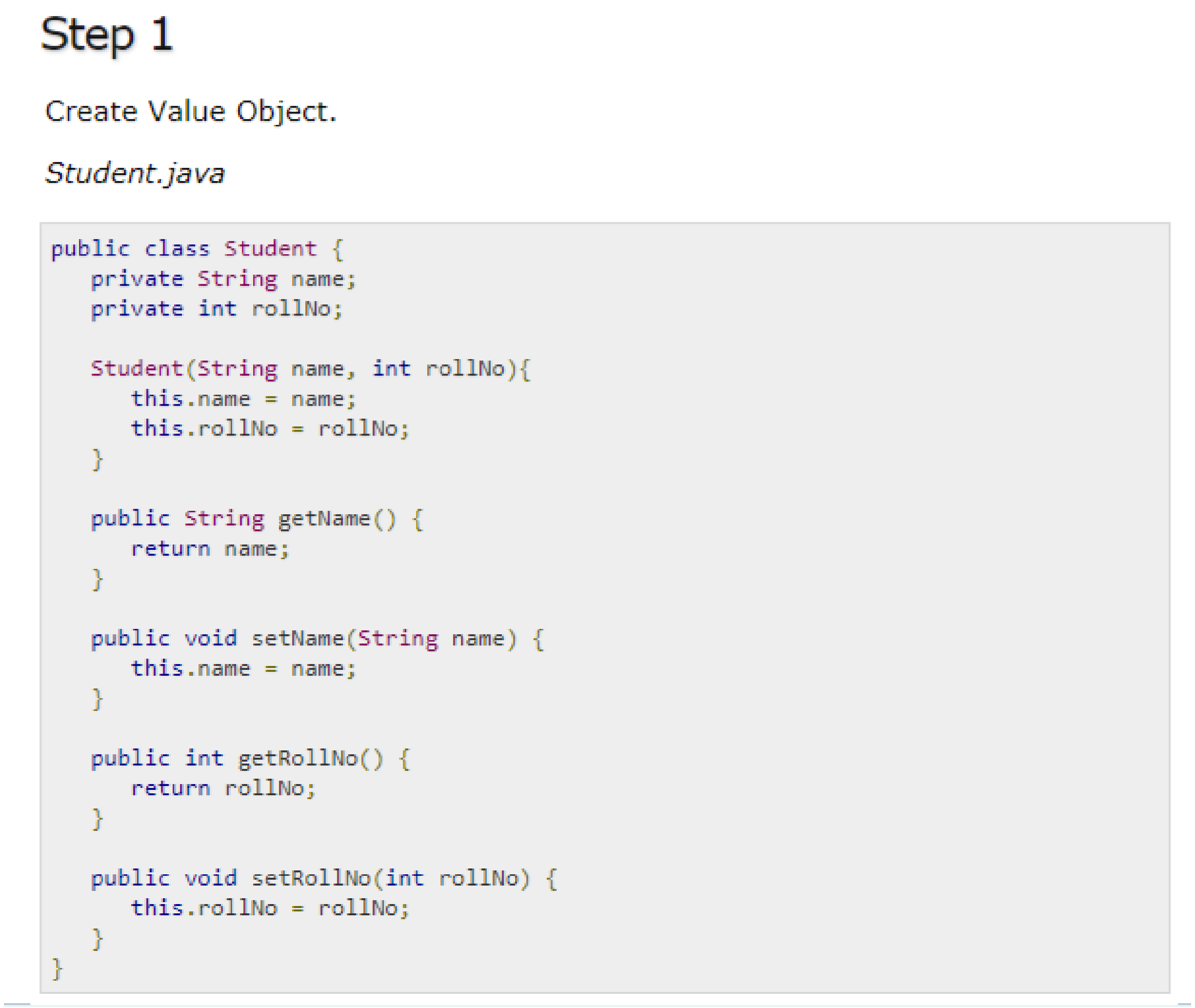


Source: https://www.tekportal.net/unified-modeling-language/

Data Access Object (DAO) pattern

- Data Access Object Pattern or DAO pattern is used to separate low level data accessing API or operations from high level business services. Following are the participants in Data Access Object Pattern.
  - Data Access Object Interface - This interface defines the standard operations to be performed on a model object(s).
  - Data Access Object concrete class - This class implements above interface. This class is responsible to get data from a data source which can be database / xml or any other storage mechanism.
  - Model Object or Value Object - This object is simple POJO containing get/set methods to store data retrieved using DAO class.

Data Access Object (DAO) pattern



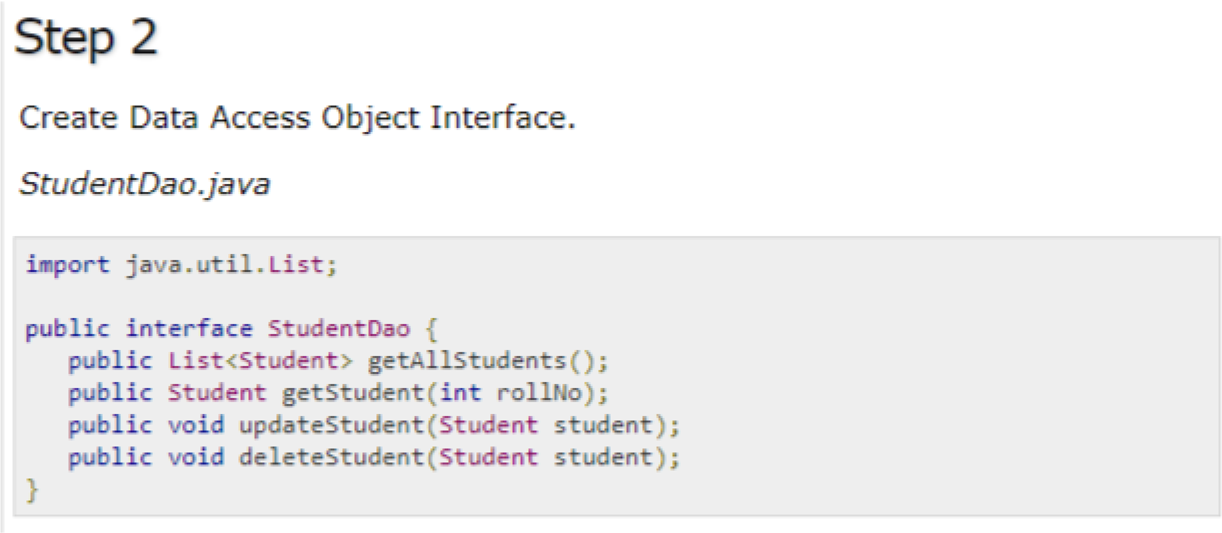
Step 1

Create Value Object.

Student.java

```
public class Student {
   private String name;
   private int rollNo;

   Student(String name, int rollNo){
      this.name = name;
      this.rollNo = rollNo;
   }

   public String getName() {
      return name;
   }

   public void setName(String name) {
      this.name = name;
   }

   public int getRollNo() {
      return rollNo;
   }

   public void setRollNo(int rollNo) {
      this.rollNo = rollNo;
   }
}
```

Data Access Object (DAO) pattern



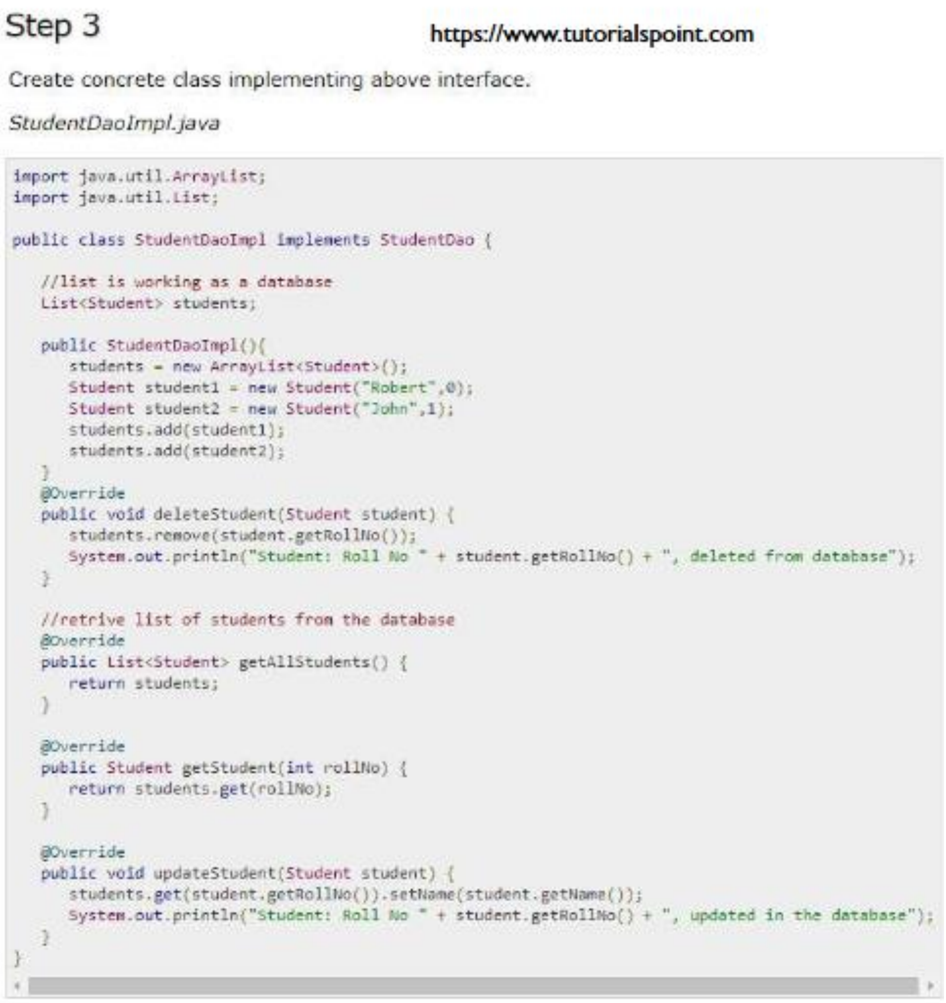
Step 2

Create Data Access Object Interface.

StudentDao.java

```
import java.util.List;

public interface StudentDao {
   public List<Student> getAllStudents();
   public Student getStudent(int rollNo);
   public void updateStudent(Student student);
   public void deleteStudent(Student student);
}
```

Data Access Object (DAO) pattern



Step 3

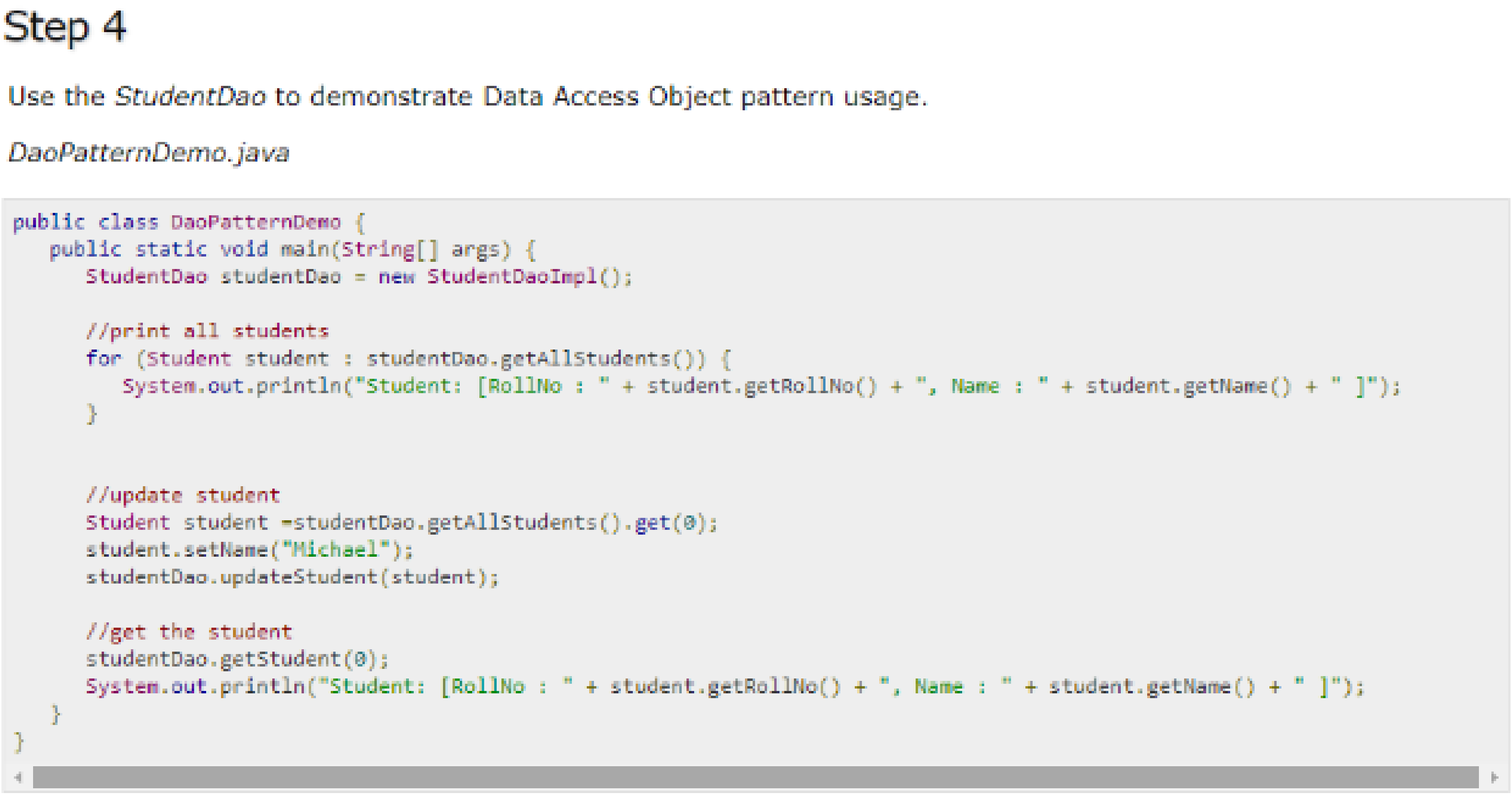
https://www.tutorialspoint.com

Create concrete class implementing above interface.

StudentDaoImpl.java

```java
import java.util.ArrayList;
import java.util.List;

public class StudentDaoImpl implements StudentDao {

   //list is working as a database
   List<Student> students;

   public StudentDaoImpl(){
      students = new ArrayList<Student>();
      Student student1 = new Student("Robert",0);
      Student student2 = new Student("John",1);
      students.add(student1);
      students.add(student2);
   }
   @Override
   public void deleteStudent(Student student) {
      students.remove(student.getRollNo());
      System.out.println("Student: Roll No " + student.getRollNo() + ", deleted from database");
   }

   //retrive list of students from the database
   @Override
   public List<Student> getAllStudents() {
      return students;
   }

   @Override
   public Student getStudent(int rollNo) {
      return students.get(rollNo);
   }

   @Override
   public void updateStudent(Student student) {
      students.get(student.getRollNo()).setName(student.getName());
      System.out.println("Student: Roll No " + student.getRollNo() + ", updated in the database");
   }
}
```

Data Access Object (DAO) pattern



Step 4

Use the *StudentDao* to demonstrate Data Access Object pattern usage.

DaoPatternDemo.java

```
public class DaoPatternDemo {
   public static void main(String[] args) {
      StudentDao studentDao = new StudentDaoImpl();

      //print all students
      for (Student student : studentDao.getAllStudents()) {
         System.out.println("Student: [RollNo : " + student.getRollNo() + ", Name : " + student.getName() + " ]");
      }


      //update student
      Student student =studentDao.getAllStudents().get(0);
      student.setName("Michael");
      studentDao.updateStudent(student);

      //get the student
      studentDao.getStudent(0);
      System.out.println("Student: [RollNo : " + student.getRollNo() + ", Name : " + student.getName() + " ]");
   }
}
```