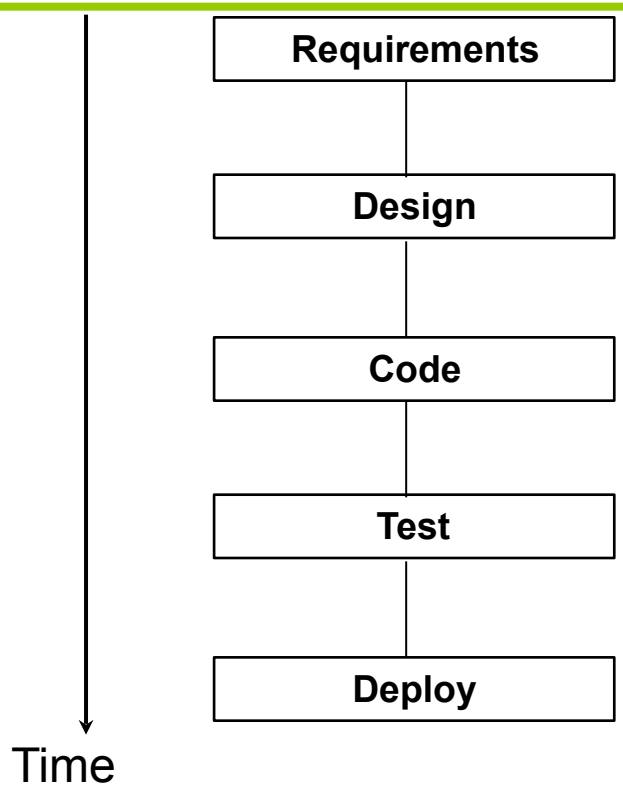
中国科学技术大学软件学院
SCHOOL OF SOFTWARE ENGINEERING OF USTC

The Unified Process

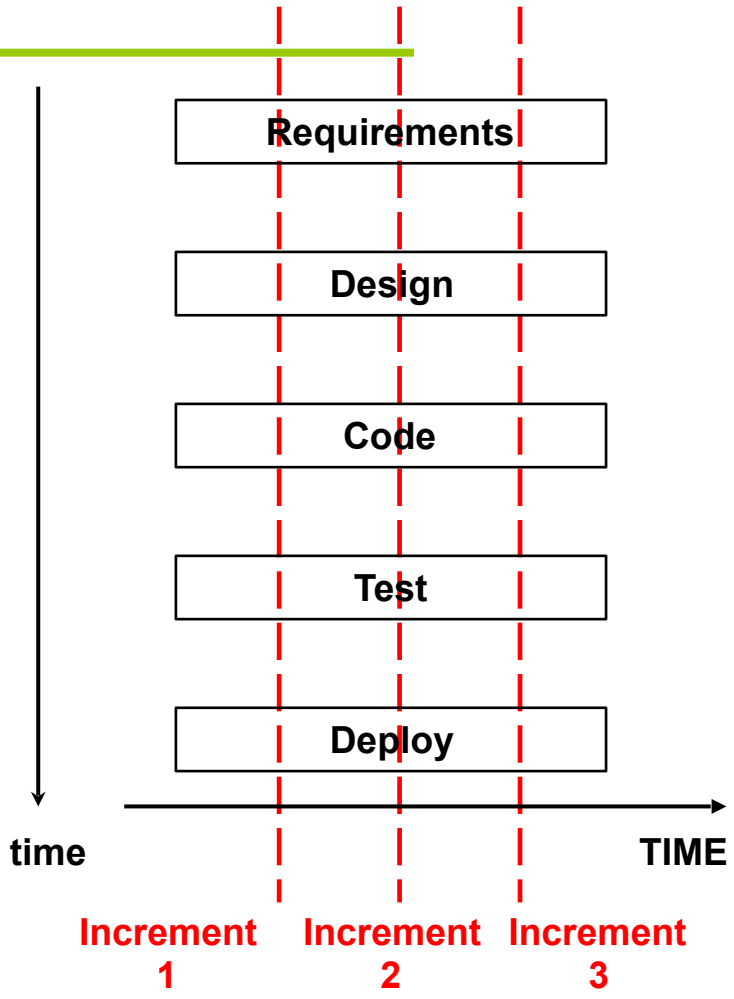
Object Interaction Modeling
and
Design Class Diagram

中国科学技术大学软件学院
孟宁

The Unified Process



The waterfall process



The unified process

The Agile Unified Process

- Before the iterations, there is a "plan and elaborate" phase
  - motivation, business needs, alternatives
  - requirements elicitation and specification
  - feasibility study
  - use cases and use case diagrams
  - requirements and use cases traceability matrix
  - draft conceptual model
  - schedule, resources, budget

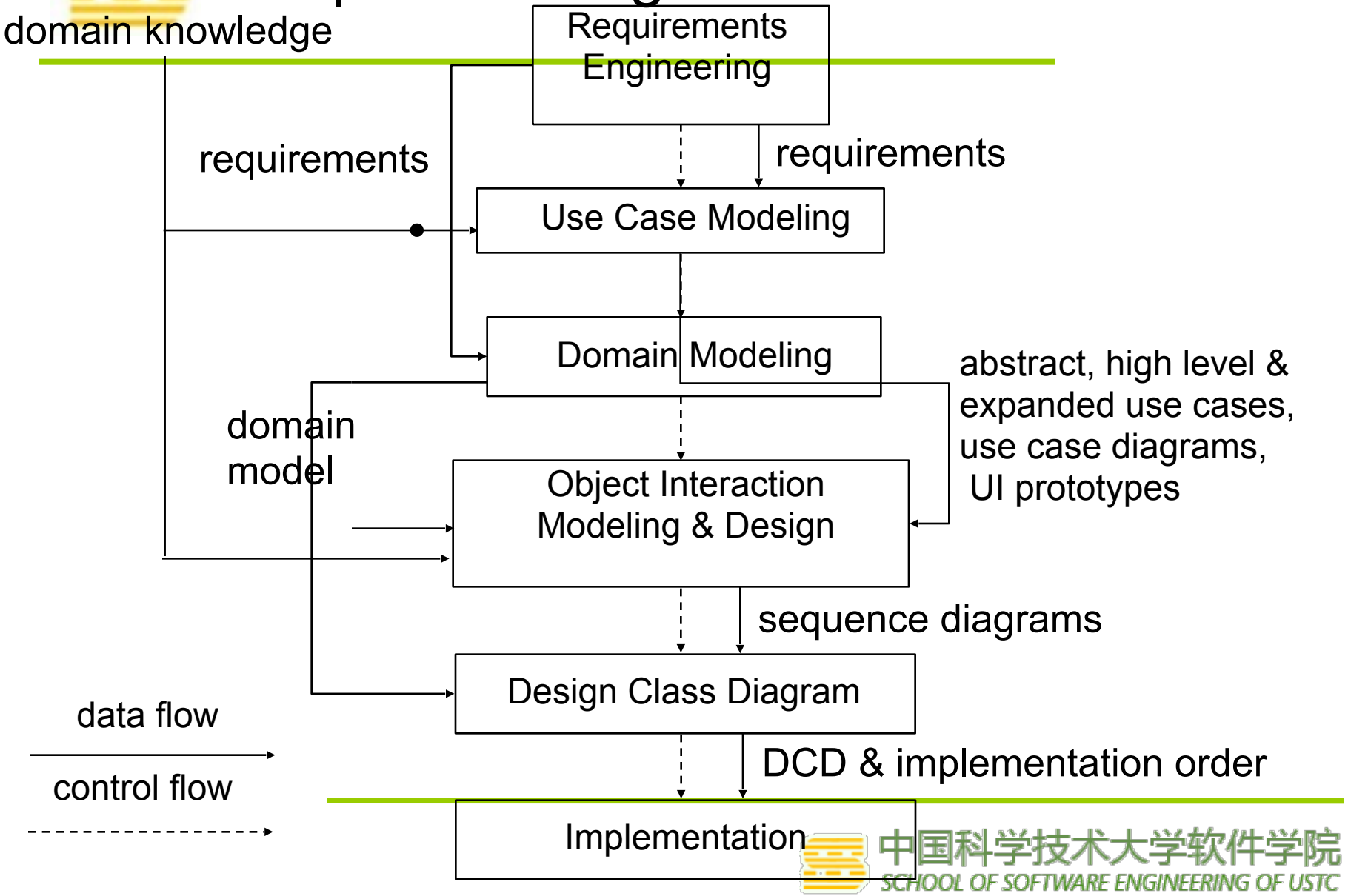
Steps of the Unified Process

1) Identifying requirements
2) Deriving use cases to satisfy the requirements
3) Allocating use cases to increments
4) Carrying out each increment

Carrying out each increment

4) Carry out each increment
   4.1) Use case modeling
   4.2) Domain modeling
   4.3) Interaction modeling
   4.4) Derive design class diagram
   4.5) Implementation and deployment

Steps of the Agile Unified Method



The Unified Process

- Use case driven
  - each iteration is driven by the use cases allocated to the iteration
- Architecture centric
  - “a system's architecture is used as a primary artifact for conceptualizing, constructing, managing, and evolving the system under development."

    --- Grady Booch
- Incremental
- Iterative

Object Interaction Modeling

- It specifies how objects interact with each other to carry out the background processing of a business process.
- Object interaction modeling is aided by the specification of scenarios and a scenario table.
- A scenario is an informal, step-by-step description of object interaction.
- A scenario table organizes the interaction into a five column table. It facilitates the translation into a sequence diagram.

Steps for Interaction Modeling

For each expanded use case do the following:

Step 1. Identify and mark nontrivial steps in the right column of the expanded use case.

A nontrivial step is one that requires background processing that goes beyond the presentation layer (i.e., the Graphical User Interface or GUI).

Step 2. For each nontrivial step, describe a scenario, i.e., how objects interact to carry out the step, beginning with the step on the left column of the expanded use case.

Step 3. If desired, convert the scenario into a scenario table.

Step 4. Convert the scenarios/scenario table into a sequence diagram.

Scenarios

- Each sentence of the scenario is a declarative sentence consisting of
  1. a subject,
  2. an action of the subject,
  3. an object that is acted upon, and
  4. possibly other objects required by the action.
- The sentences are arranged in a scenario table to
  - facilitate scenario description and
  - translation into sequence diagrams.

Example

UC1: Checkout Document

Actor: Patron	System: LIS
	0) System displays the main GUI.
1) TUCBW patron clicks the "Checkout Document" button on the main GUI.	2) The system displays the Checkout GUI.
3) The patron enters the call numbers of documents to be checked out and clicks the "Submit" button.	* 4) The system displays a checkout message showing the details.
5) TUCEW the patron sees the checkout message.	

nontrivial step

Scenario Description

4) Checkout GUI checks out the documents with the checkout controller using the document call numbers.
4.1) Checkout controller creates a blank msg.
4.2) For each document call number,
4.2.1) The checkout controller gets the document from the database manager (DBMgr) using the document call number.
4.2.2) DBMgr returns the document d to the checkout controller.
4.2.3) If the document exists (i.e., d!=null)
4.2.3.1) the checkout controller checks if the document is available (for check out).
4.2.3.2) If the document is available for check out,
4.2.3.2.1) the checkout controller creates a Loan object for the document d,
4.2.3.2.2) the checkout controller sets the document as not available,
4.2.3.2.3) the checkout controller saves the Loan object with DBMgr,
4.2.3.4) the checkout controller saves document d with the DBMgr,
4.2.3.4.5) the checkout controller writes "checkout successful" to msg.
4.2.3.3) else,
4.2.3.3.1) the checkout controller writes "document not available" to msg.
4.2.4) else
4.2.4.1) the checkout controller writes "document not found" to msg.
4.3) The checkout controller returns msg to the Checkout GUI.
4.4) The system (Checkout GUI) displays msg to patron.

Could be simplified, e.g., reducing the number of steps.

	Subject	Action of Subject	Other Data/Objects	Object Acted Upon
4)	Checkout GUI	checks out	call numbers	checkout controller
4.1)	checkout controller	creates		msg
4.2)	For each document call number			
4.2.1)	checkout controller	gets document	call number	DBMgr
4.2.2)	DBMgr	returns	document d	checkout controller
4.2.3)	If document exists (d!=null)			
4.2.3.1)	checkout controller	checks is available		document
4.2.3.2)	If document is available			
4.2.3.2.1)	checkout controller	creates		Loan object
4.2.3.2.2)	checkout controller		false	
4.2.3.2.3)	checkout controller	saves		
4.2.3.2.4)	checkout controller	saves		DBMgr
4.2.3.2.5)	checkout controller		"checkout successful"	msg
4.2.3.3)	else			
4.2.3.3.1)	checkout controller	appends	"document not available"	msg
4.2.4)	else			
4.2.4.1)	checkout controller	appends	"document not found"	msg
4.3)	checkout controller	returns	msg	Checkout GUI
4.4)	Checkout GUI	displays	msg	Patron

Could be simplified, e.g., reducing the number of steps.

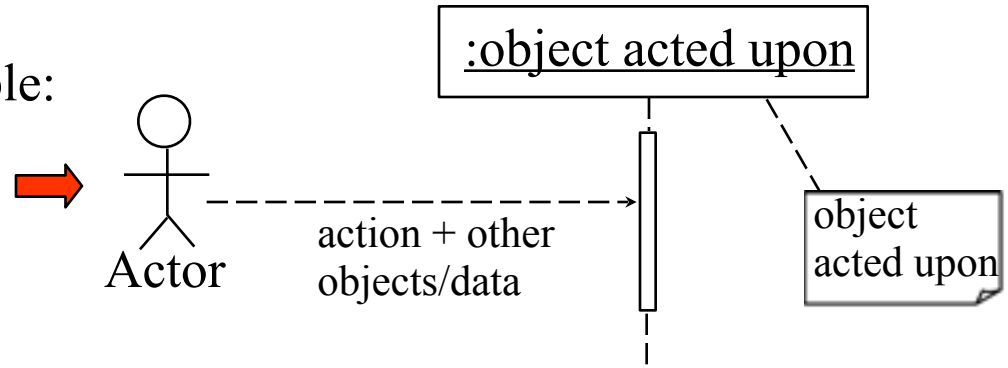
Guidelines for Scenario Construction

- Keep it simple and stupid --- leave details to coding.
- Specify the normal scenario first (i.e., assume everything will go as expected)
- Augment the normal scenario with alternative flows if needed
- Leave exception handling to the code
- It is sometimes desirable to construct a prototype for the normal scenarios to validate the design idea

From Scenario Table to Sequence Diagram

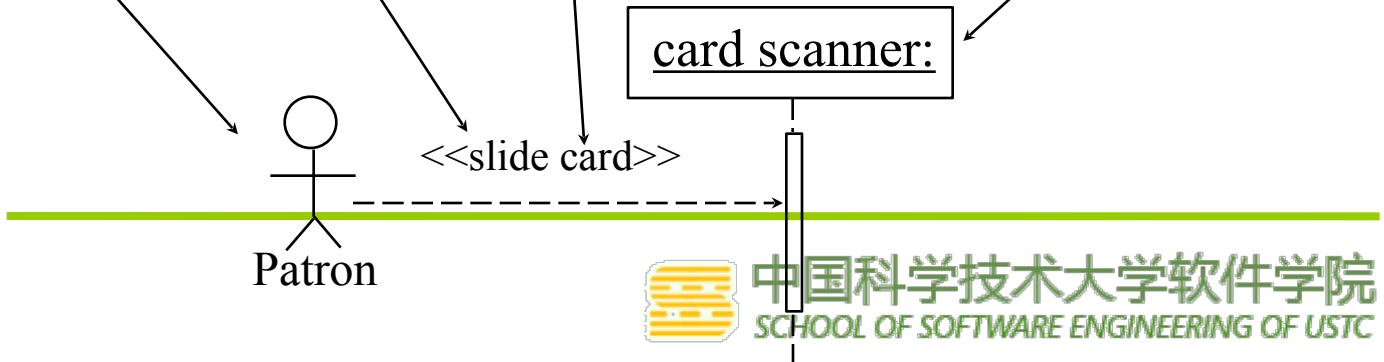
for each line of the scenario table:

Case 1: subject is an Actor



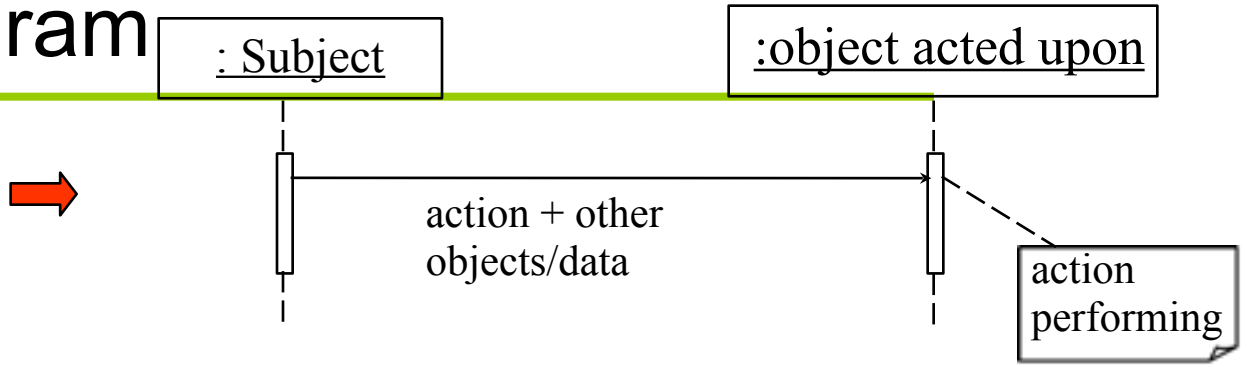
Example:

	subject	action/message	other objects/data	object acted
1	patron	slide	card	through card scanner



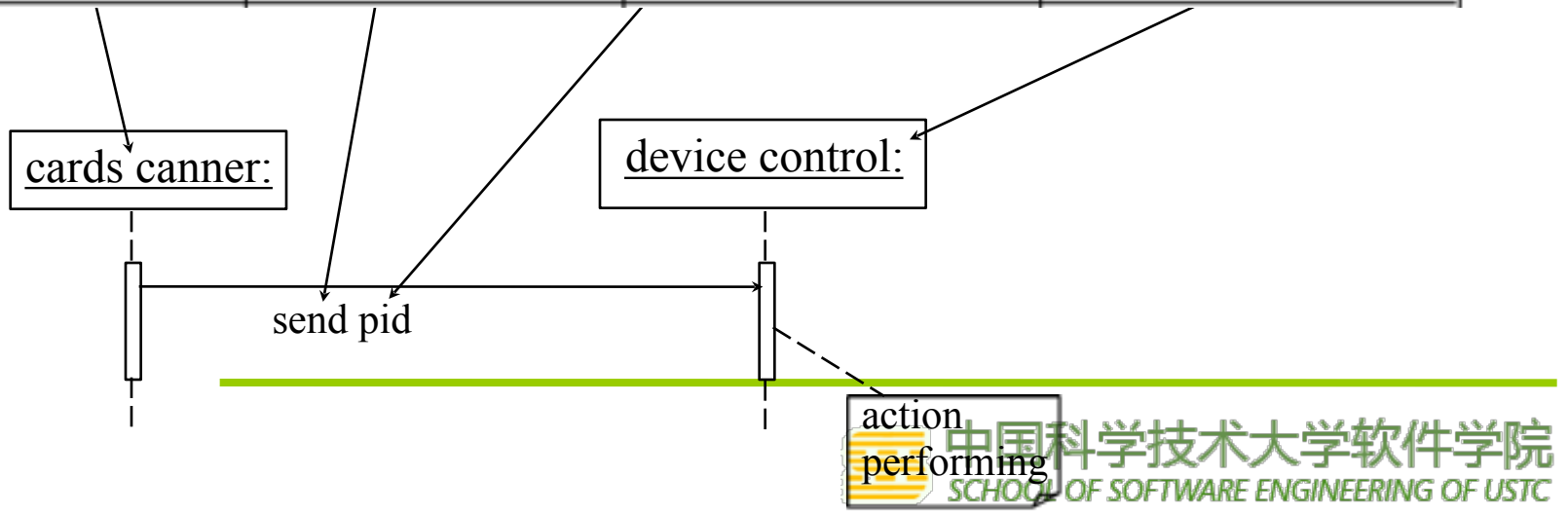
From Scenario Table to Sequence Diagram

Case 2: subject is an object



Example

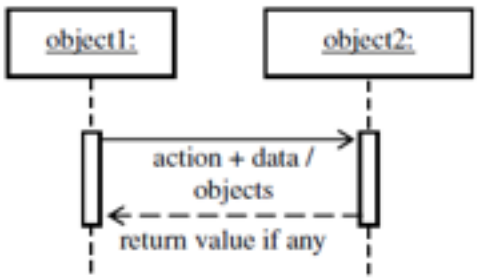
	subject	action/message	other objects/data	object acted
1	patron	slide	card	through card scanner
2	card scanner	read	patron id (pid)	from card
3	card scanner	send	pid	to device control



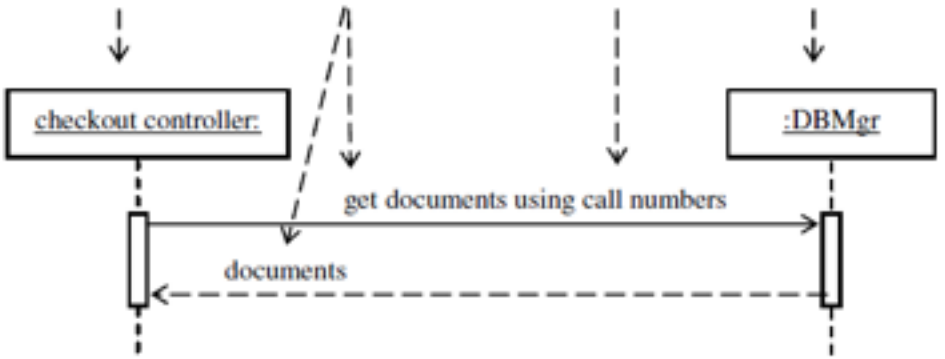
From Scenario Table to Sequence Diagram

Case 3: both subject and object acted upon are objects

Subject	Subject Action	Data / Objects	Object Acted
object1	action	data / objects	object2



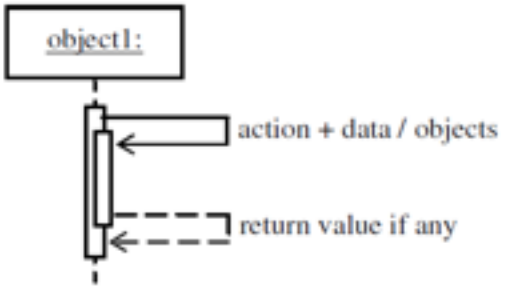
Subject	Subject Action	Other Data / Objects	Object Acted Upon
checkout controller	gets documents	call numbers	DBMgr



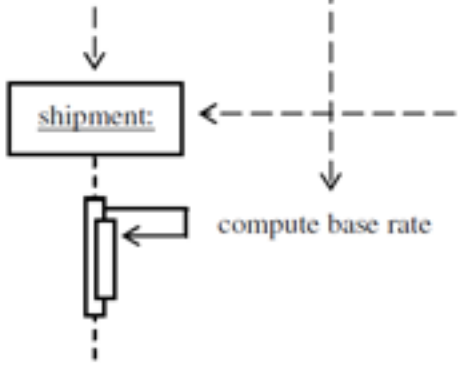
From Scenario Table to Sequence Diagram

Case 4: both subject and object acted upon are the same (a special case of case 3)

Subject	Subject Action	Data / Objects	Object Acted
object1	action	data / objects	object1



Subject	Subject Action	Other Data / Objects	Object Acted Upon
shipment	compute base rate		shipment

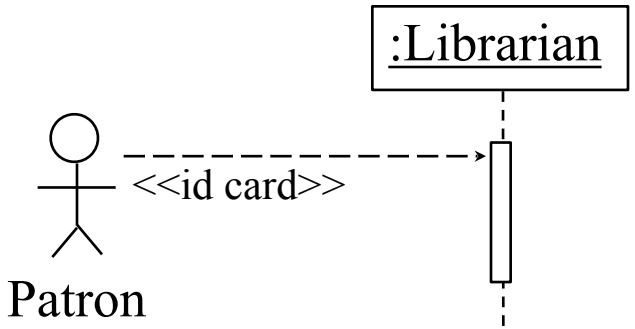


Difference of Analysis and Design

Analysis	Design
♦ application problem-oriented	♦ software solution-oriented
♦ modeling application domain	♦ modeling the software system
♦ describes what the world is	♦ prescribes a software solution
♦ project-oriented decisions	♦ system-oriented decisions
♦ should allow multiple design alternatives	♦ usually reduces implementation choices

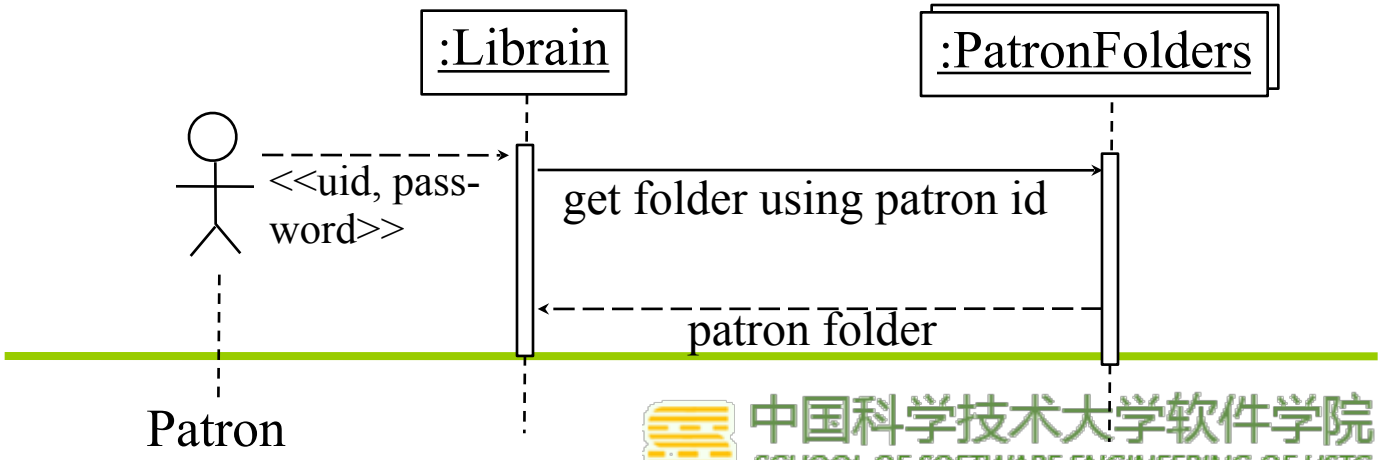
Modeling a Manual Library System

Patron presents id card to librarian.

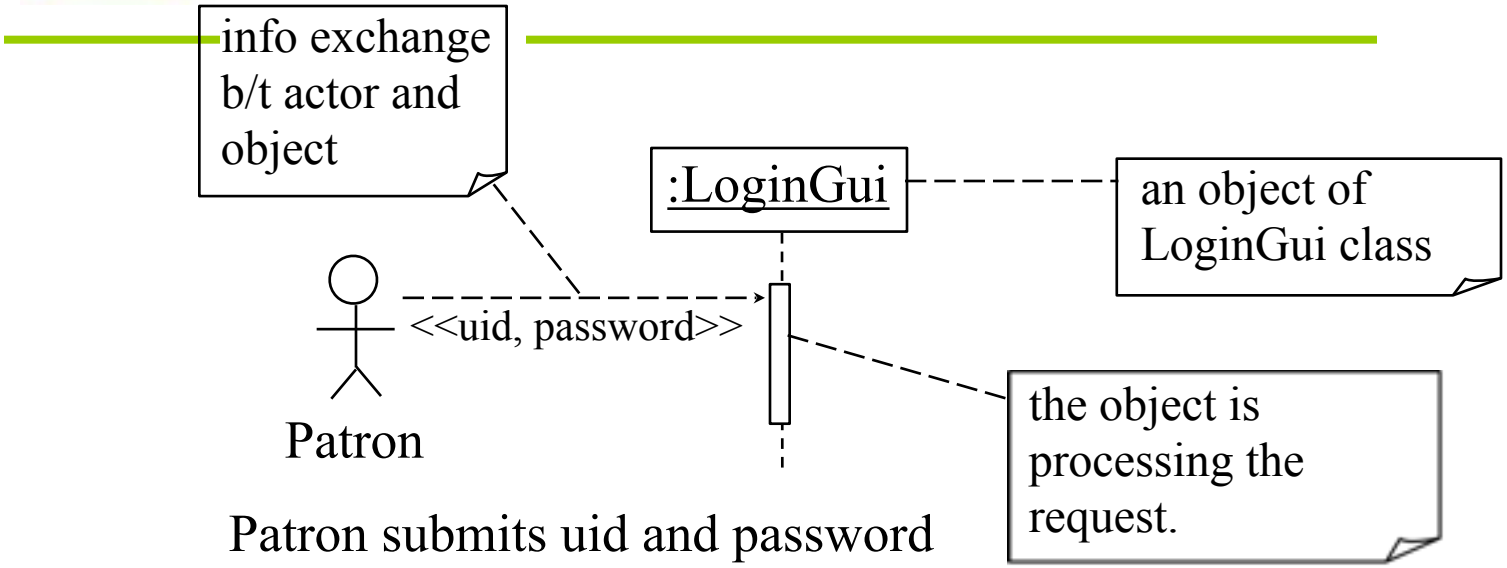


These are analysis sequence diagrams that model an existing system.

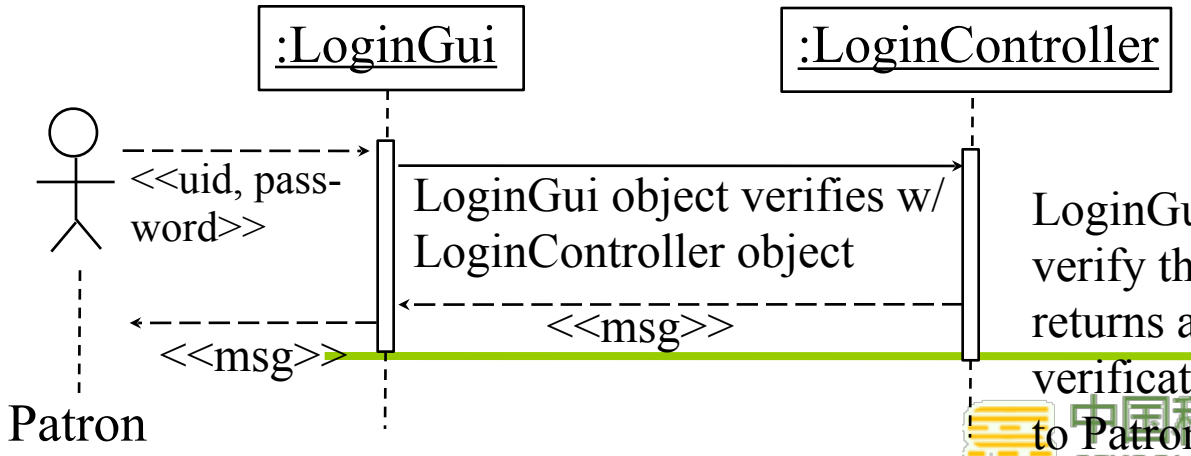
Librarian pulls out patron's folder using id number.



Analysis/Informal Sequence Diagram

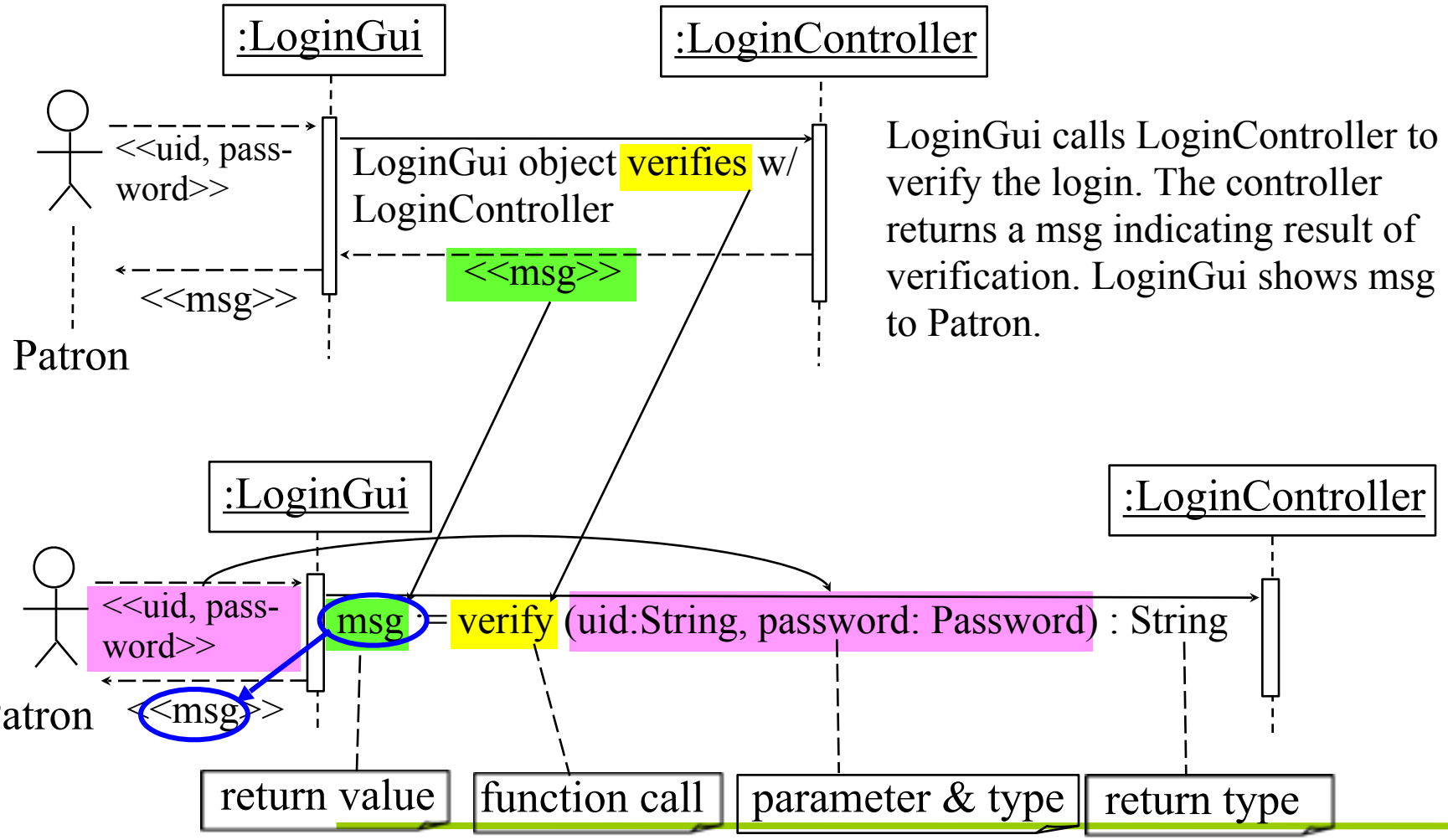


Patron submits uid and password to LoginGui.

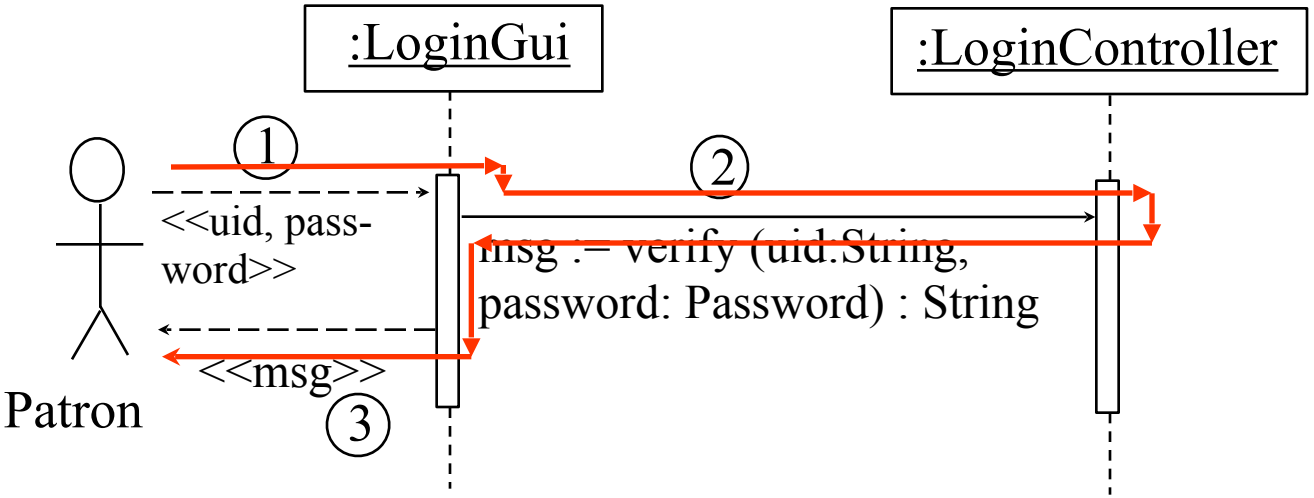


LoginGui calls LoginController to verify the login. The controller returns a msg indicating result of verification. LoginGui shows msg to Patron.

From Analysis to Design



Sequence Diagram: Flow of Control

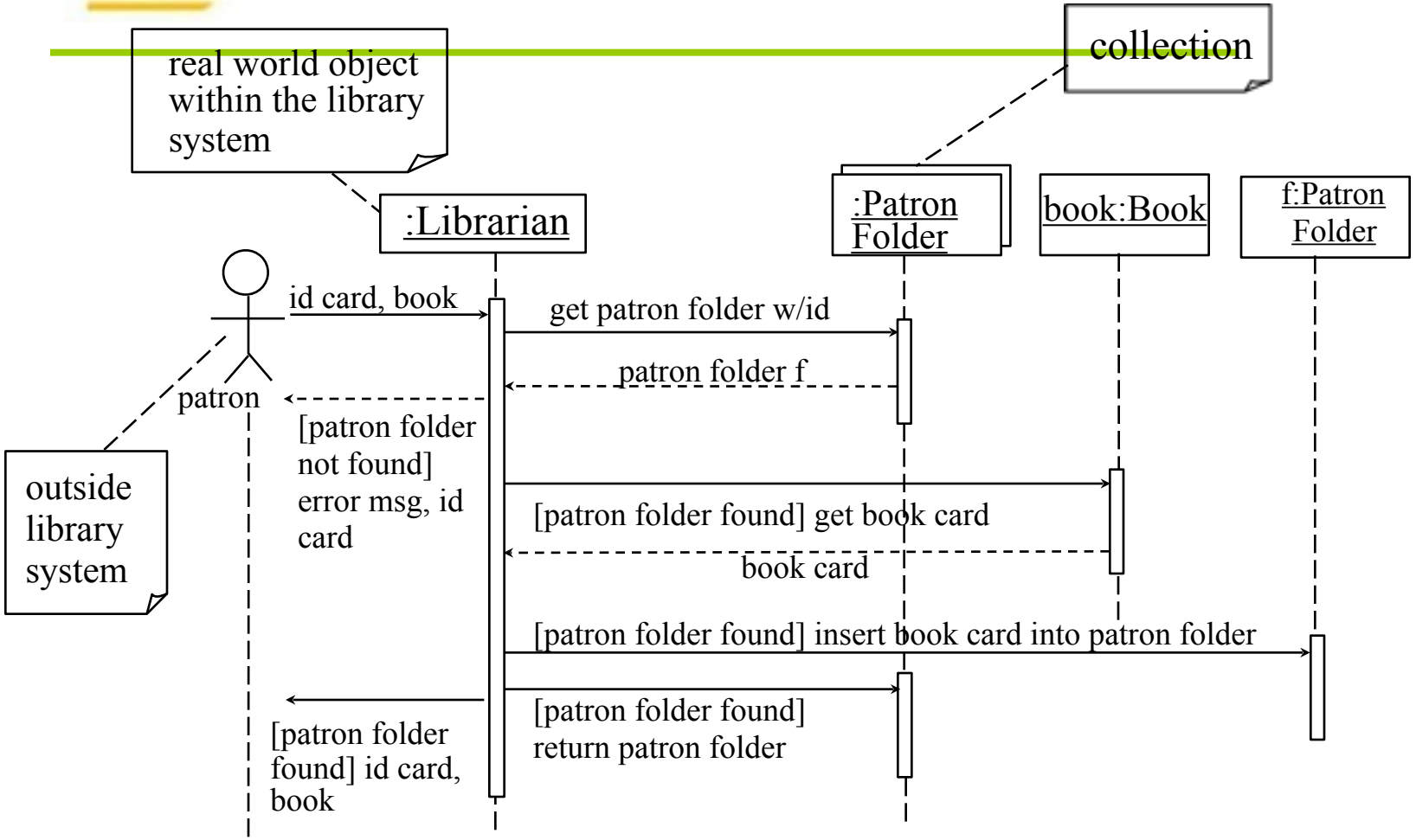


(1) patron submits uid and password to LoginGui object

(2) LoginGui object calls the verify function of a LoginController object. The called function returns msg of type String.

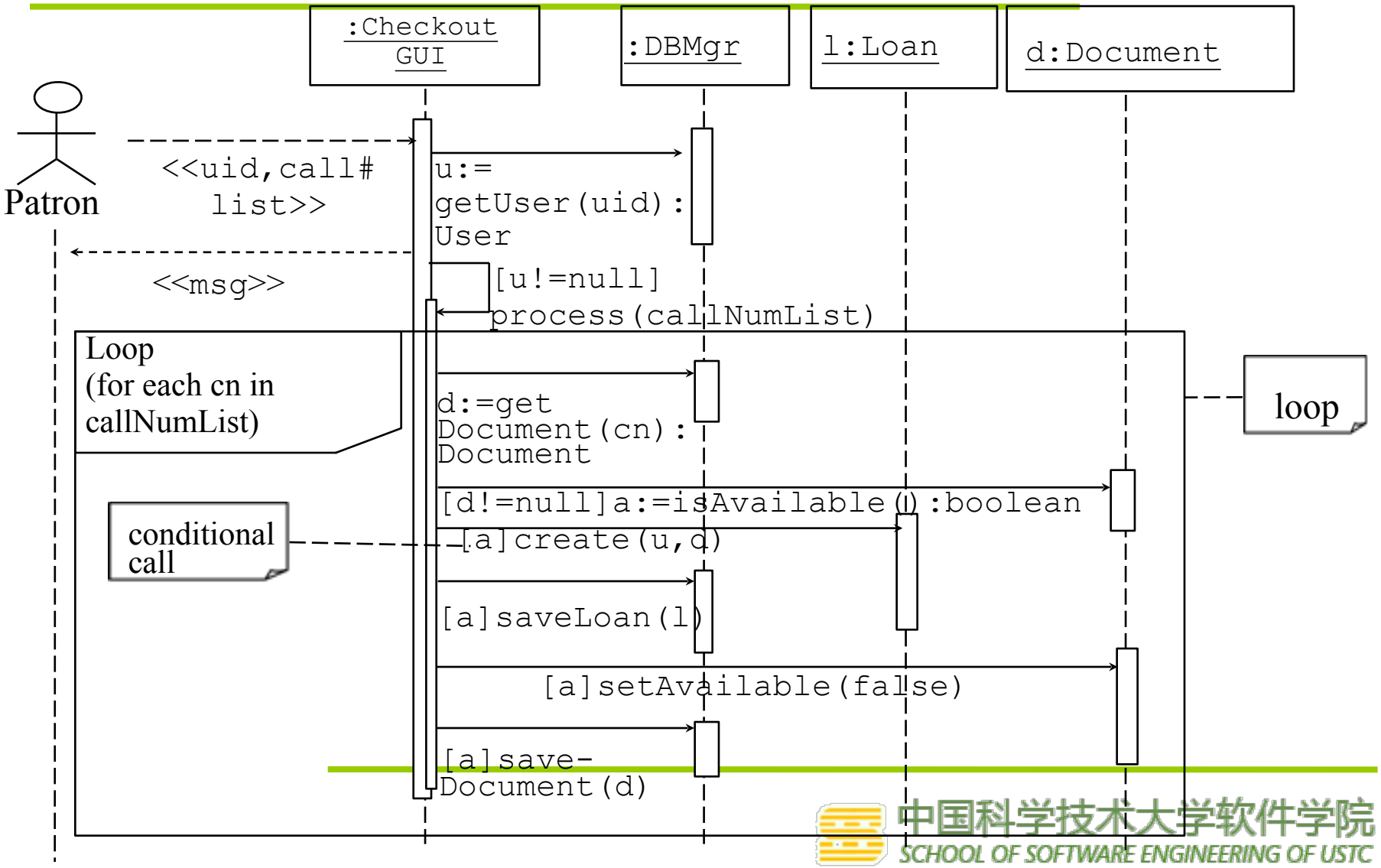
(3) The LoginGui object shows msg to patron.

An Analysis Sequence Diagram



This an analysis sequence diagram models the current, manual operation, little design decision is made.

A Design Sequence Diagram



Deriving Design Class Diagram

- Design class diagram (DCD) is a structural diagram.
- It shows the classes, their attributes and operations, and relationships between the classes. It may also show the design patterns used.
- It is used as a basis for implementation, testing, and maintenance.
- It should contain only classes appear in the sequence diagrams, and a few classes from the domain model.

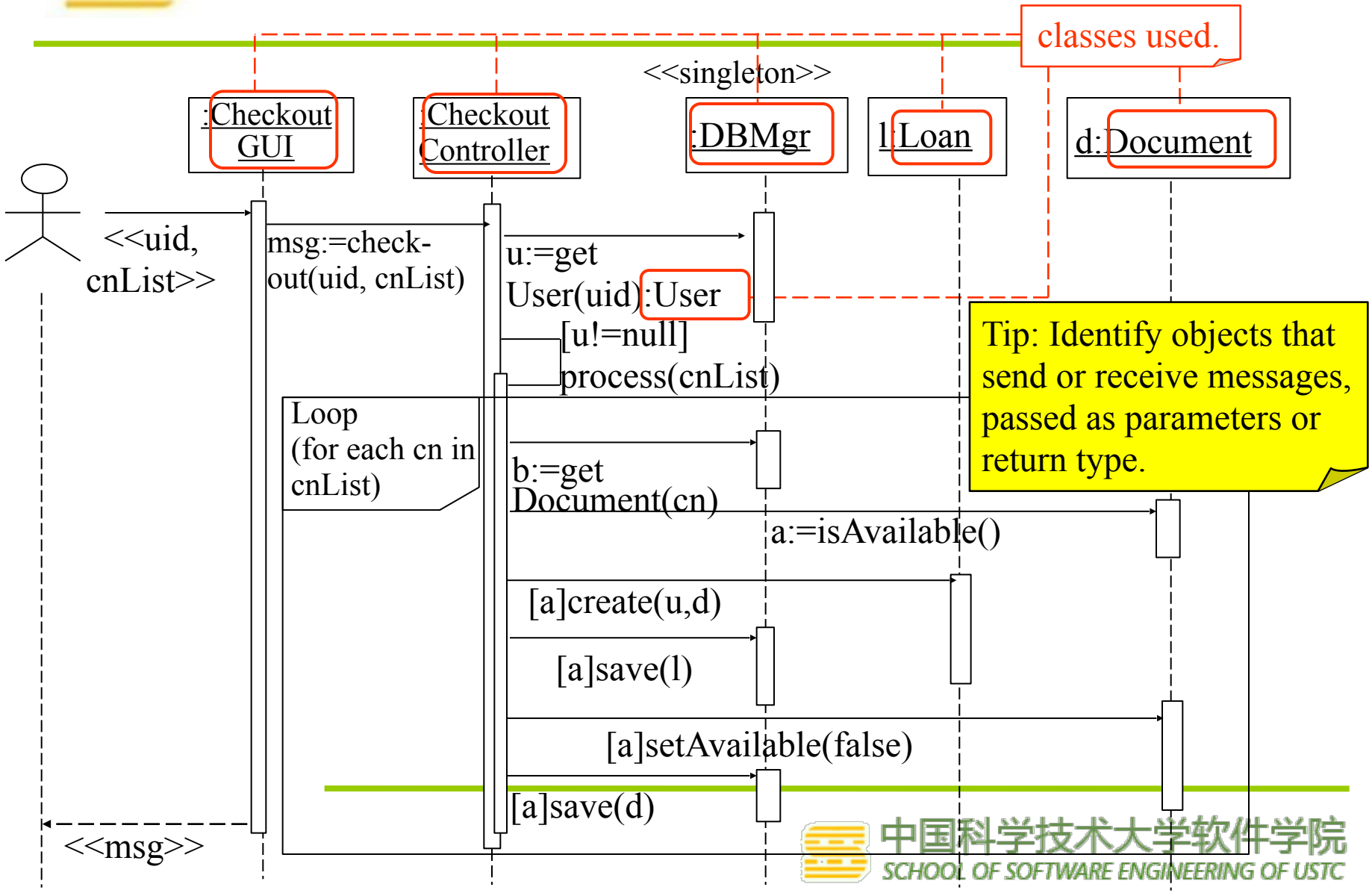
Deriving Design Class Diagram

- It is derived from the domain model (DM) and the sequence diagrams:
  - the domain model provides the attributes and some relationships
  - the sequence diagrams determines the classes and methods, and provides dependence relationships
- DCD may contain design classes like controller, command, GUI classes. Domain model only contains application objects (classes).
- DCD must be carefully specified. DM is more liberal.

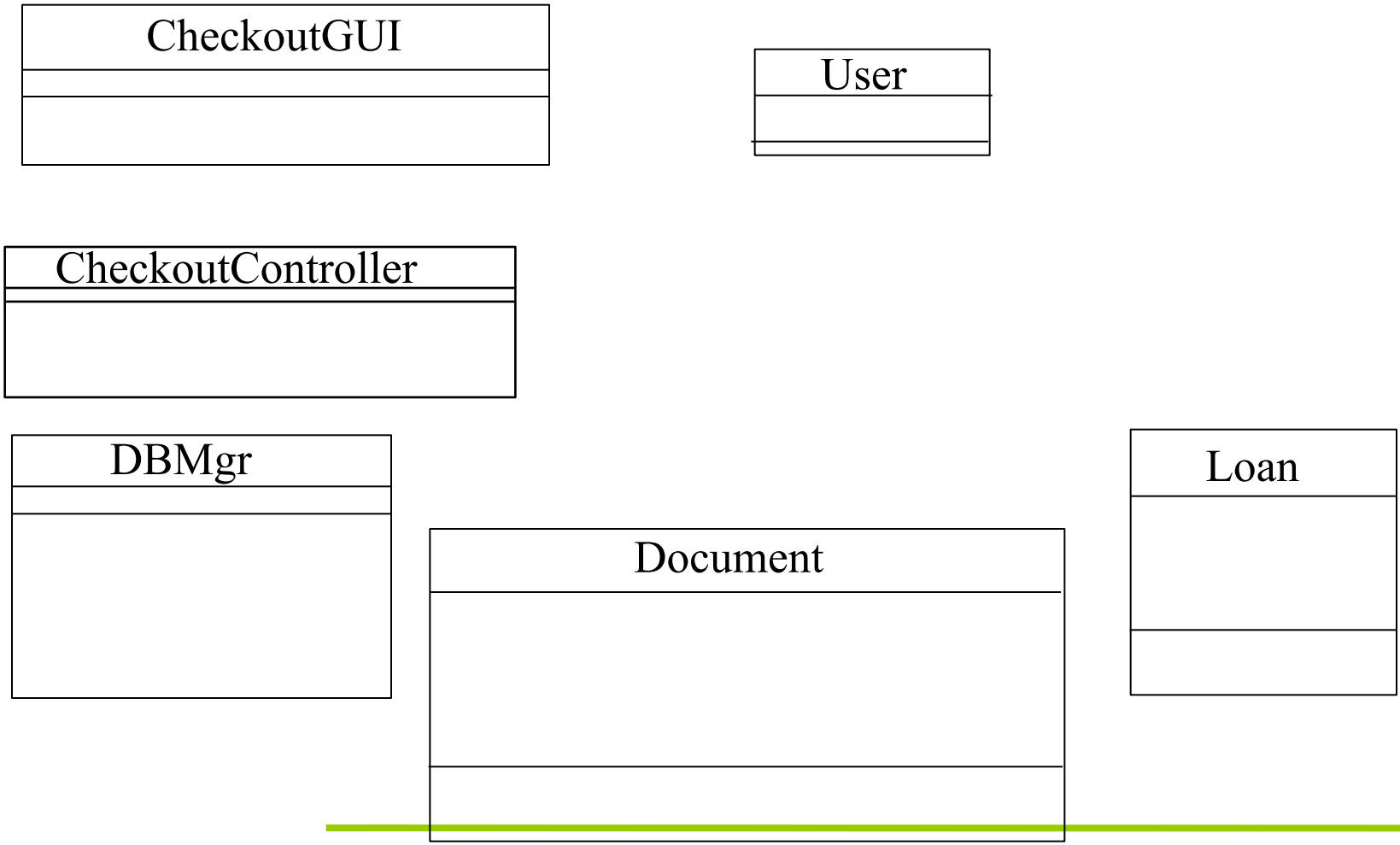
Steps for Deriving DCD

1) Identify all classes used in each of the sequence diagrams and put them down in the DCD:
   - classes of objects that send or receive messages
   - classes of objects that are passed as parameters or return types/values

Identify Classes Used in Sequence Diagrams



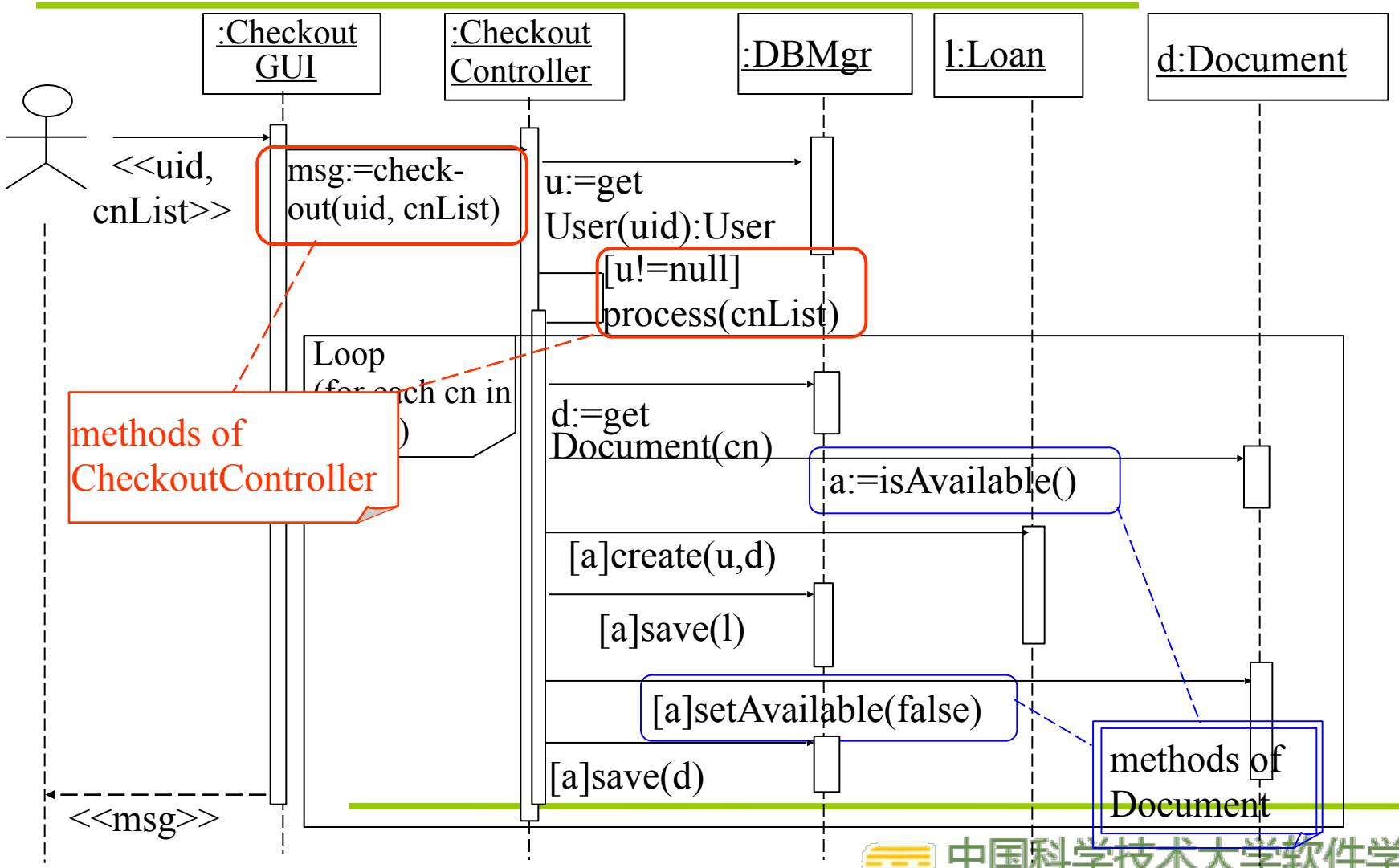
Classes Identified



Steps for Deriving DCD

2) Identify methods belonging to each class and fill them in the DCD:
   - methods are identified by looking for messages that label an incoming edge of the object
   - the sequence diagram may also provide detailed information about the parameters, their types, and return types

Identify Methods



Fill In Identified Methods

CheckoutGUI

User

CheckoutController
checkout(uid,cnList) process(cn:String[])

<<singleton>> DBMgr
getUser(uid) getDocument(callNo) saveLoan(loan) saveDocument(book)

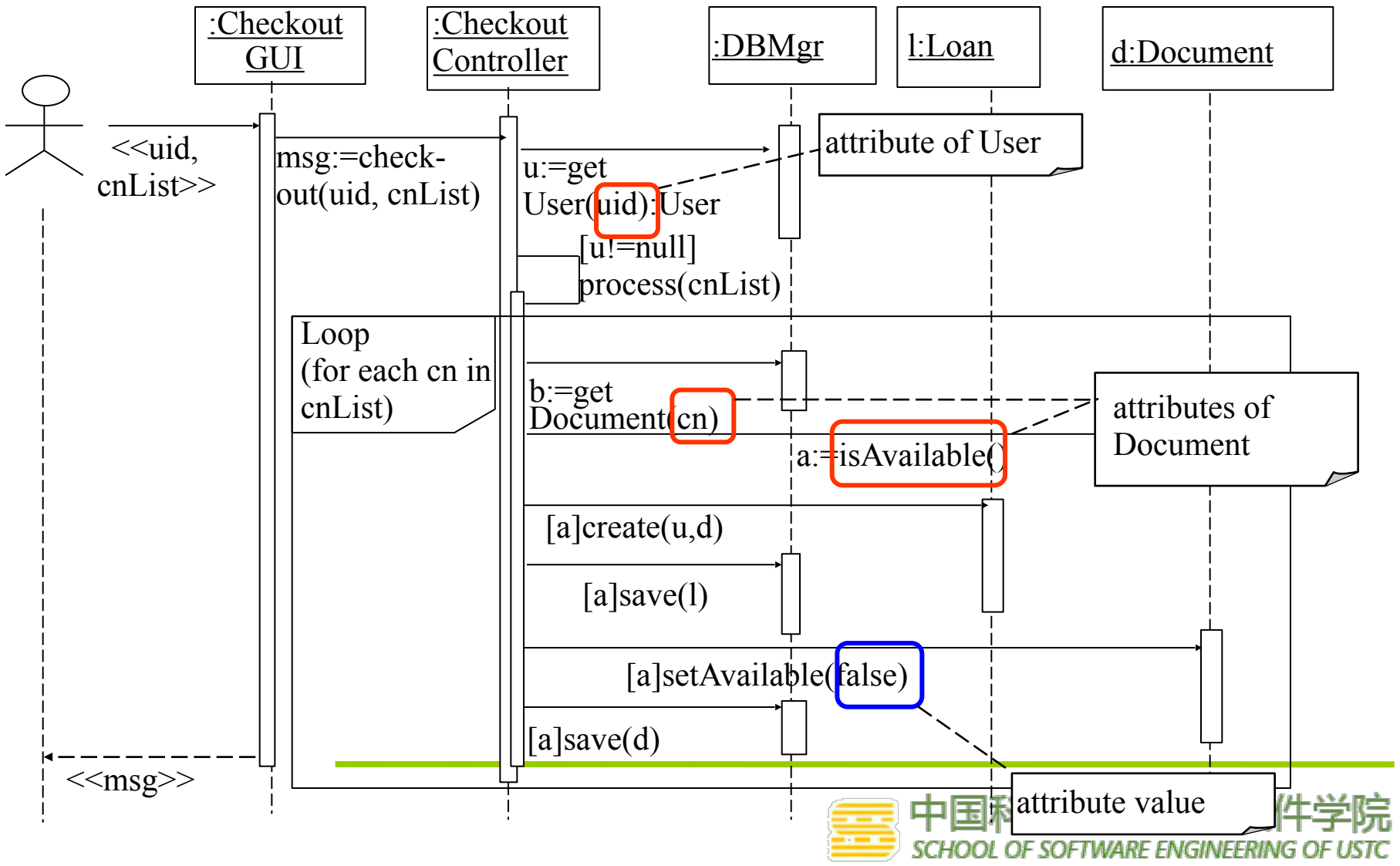
Document
isAvailable() : boolean setAvailable(a:boolean)

Loan
create(u:User, d:Document)

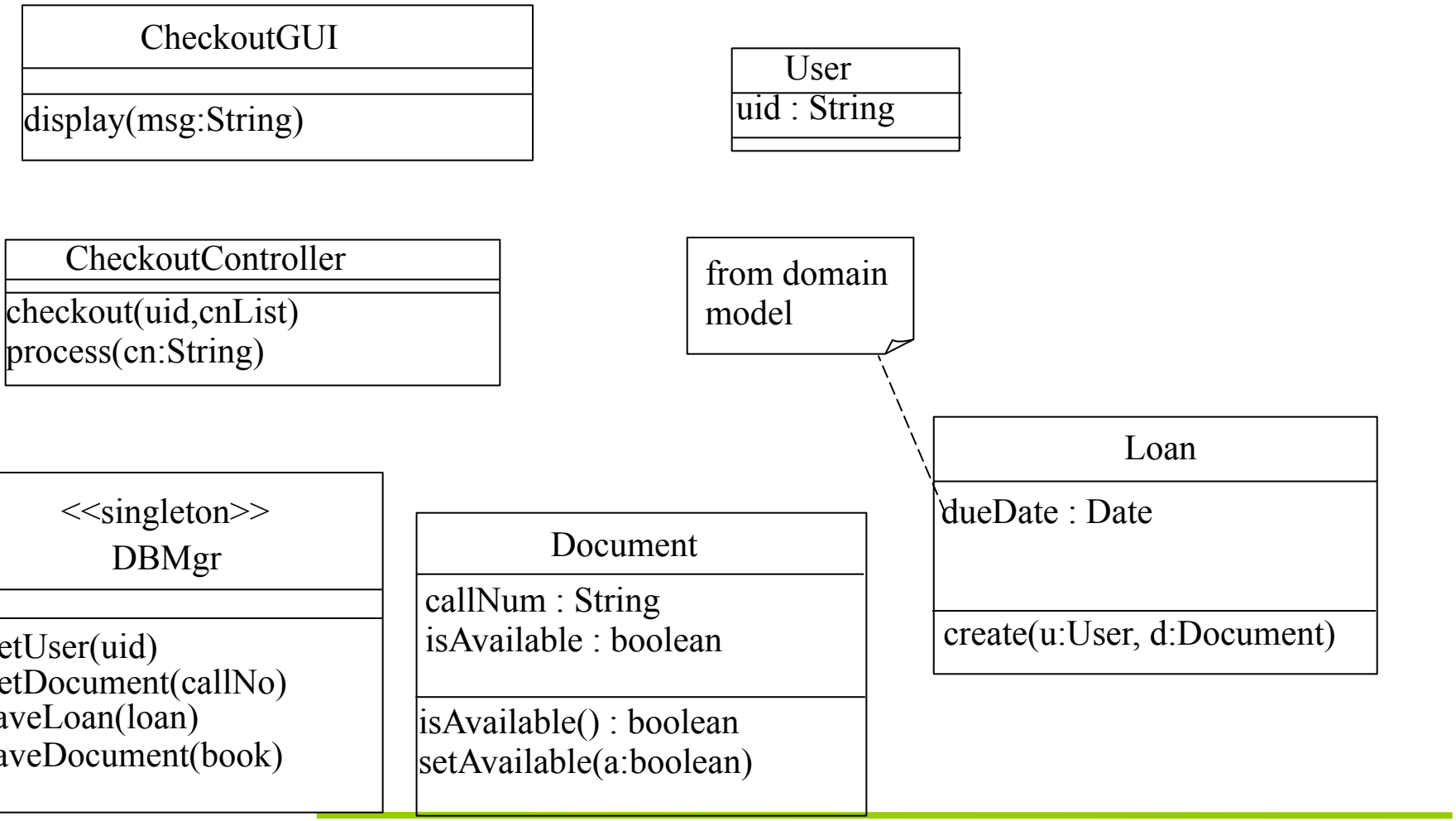
Steps for Deriving DCD

3) Identify and fill in attributes from sequence diagrams and domain model:
   - attributes are not objects and have only scalar types
   - attributes may be used to get objects
   - attributes may be identified from getX() and setX(...) methods
   - needed attributes may also be found in the domain model

Identify Attributes



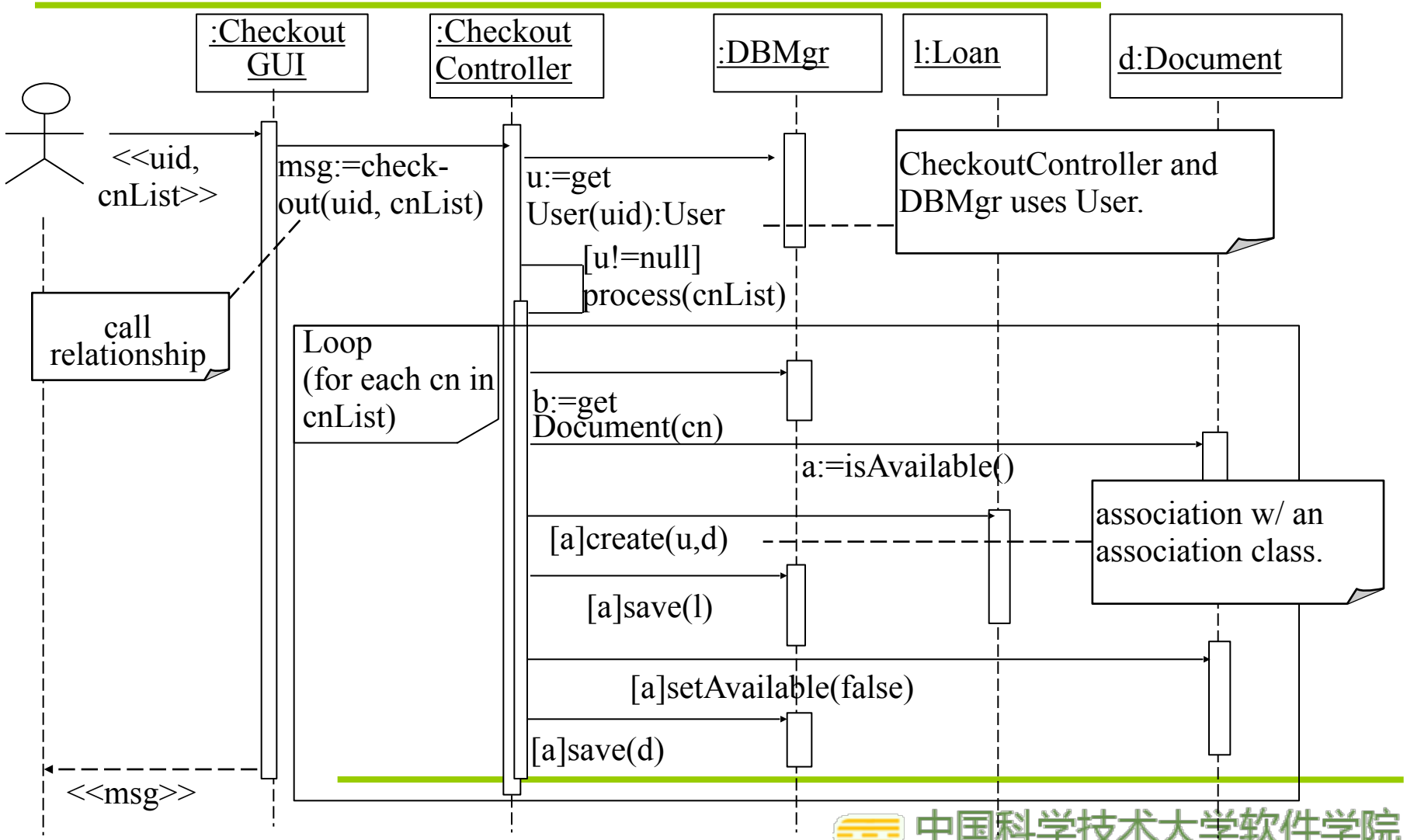
Fill In Attributes



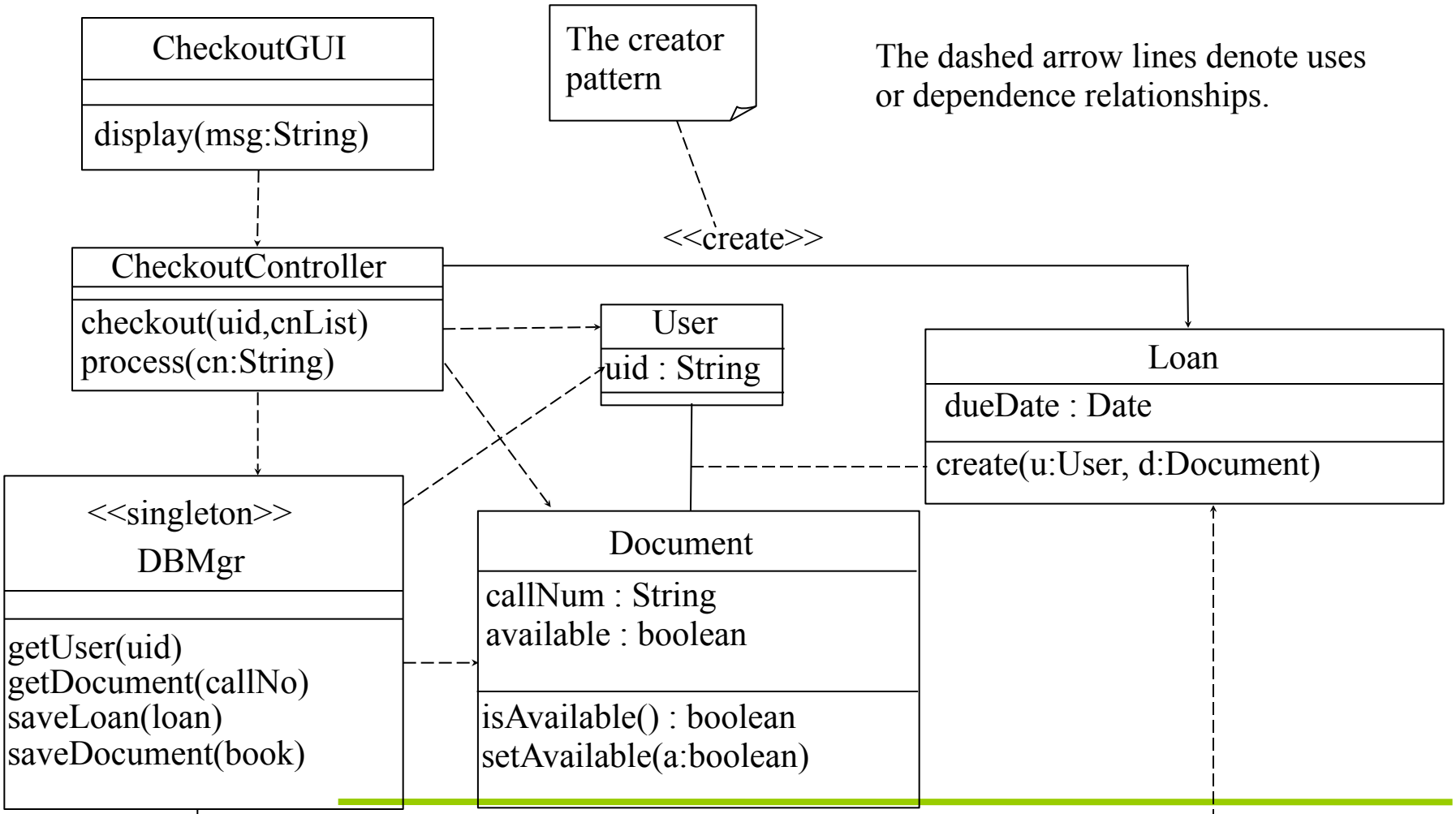
Steps for Deriving DCD

4) Identify and fill in relationships from sequence diagram and domain model:
   - arrows from one object to another is a call and hence it is a dependence relationship
   - objects passed as parameters or return type/value is an association or uses relationship
   - two or more objects passed to a constructor may indicate association and an association class
   - the domain model may contain useful relationships as well

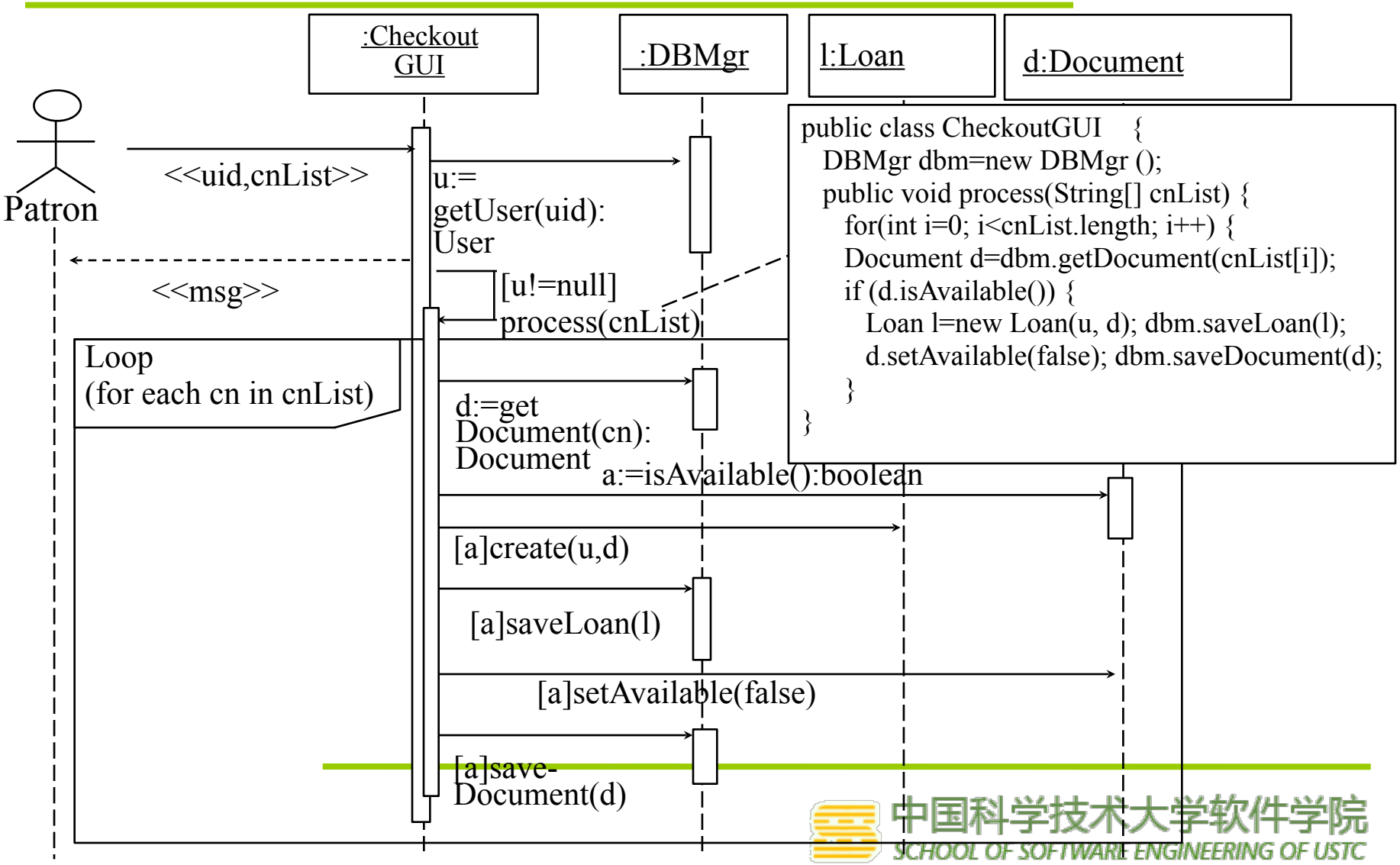
Identify Relationships



Fill In Relationships



From Sequence Diagram to Implementation



谢谢大家！

References
Dr. David Kung University of Texas Arlington May 2010