

CSc 322 - Software Engineering

Lab – Spring 2017

Instructor: Arun Adiththan, email: arun.cuny@gmail.com

Identifying the Requirements from Problem Statements

February 17, 2017

Introduction

Requirements identification is the first step of any software development project. Until the requirements of a client have been clearly identified, and verified, no other task (design, coding, testing) could begin. Usually business analysts having domain knowledge on the subject matter discuss with clients and decide what features are to be implemented.

In this session we will learn how to identify functional and non-functional requirements from a given problem statement. Functional and non-functional requirements are the primary components of a Software Requirements Specification.

Requirements: Requirements specify how the target system should behave. It specifies what to do, but not how to do. Requirements engineering refers to the process of understanding what a customer expects from the system to be developed, and to document them in a standard and easily readable and understandable format. This documentation will serve as reference for the subsequent design, implementation and verification of the system.

It is necessary and important that before we start planning, design and implementation of the software system for our client, we are clear about it's requirements. If we don't have a clear vision of what is to be developed and what all features are expected, there would be serious problems, and customer dissatisfaction as well.

Characteristics of Requirements: Requirements gathered for any new system to be developed should exhibit the following three properties:

- *Unambiguity:* There should not be any ambiguity what a system to be developed should do. For example, consider you are developing a web application for your client. The client requires that enough number of people should be able to access the application simultaneously. What's the "enough number of people"? That could mean 10 to you, but, perhaps, 100 to the client. There's an ambiguity.
- *Consistency:* To illustrate this, consider the automation of a nuclear plant. Suppose one of the clients say that if the radiation level inside the plant exceeds R1, all reactors should be shut down. However, another person from the client side suggests that the threshold radiation level should be R2. Thus, there is an inconsistency between the two end users regarding what they consider as threshold level of radiation.
- *Completeness:* A particular requirement for a system should specify what the system should do and also what it should not. For example, consider a software to be developed for ATM. If a customer enters an amount greater than the maximum permissible withdrawal amount, the ATM should display an error message, and it should not dispense any cash.

Categorization of Requirements: Based on the target audience or subject matter, requirements can be classified into different types, as stated below:

- *User requirements:* They are written in natural language so that both customers can verify their requirements have been correctly identified
- *System requirements:* They are written involving technical terms and/or specifications, and are meant for the development or testing teams

Requirements can be classified into two groups based on what they describe:

- *Functional requirements (FRs):* These describe the functionality of a system – how a system should react to a particular set of inputs and what should be the corresponding output.
- *Non-functional requirements (NFRs):* They are not directly related what functionalities are expected from the system. However, NFRs could typically define how the system should behave under certain situations.

Non-functional requirements could be further classified into different types like:

- *Product requirements:* For example, a specification that the web application should use only plain HTML, and no frames
- *Performance requirements:* For example, the system should remain available 24x7

Example Problem Statement – Library Information System (LIS) for an Institute

LIS will enable the members to borrow a book (or return it) with ease while sitting at his desk/chamber. The system also enables a member to extend the date of his borrowing if no other booking for that particular book has been made. For the library staff, this system aids them to easily handle day-to-day book transactions. The librarian, who has administrative privileges and complete control over the system, can enter a new record into the system when a new book has been purchased, or remove a record in case any book is taken off the shelf. Any non-member is free to use this system to browse/search books online. However, issuing or returning books is restricted to valid users (members) of LIS only.

The final deliverable would a web application, which should run only within the institute LAN. Although this reduces security risk of the software to a large extent, care should be taken no confidential information (eg., passwords) is stored in plain text.

Identification of Functional Requirements

- *New user registration:* Any member of the institute who wishes to avail the facilities of the library has to register himself with the Library Information System. On successful registration, a user ID and password would be provided to the member. He has to use this credentials for any future transaction in LIS.
- *Search book:* Any member of LIS can avail this facility to check whether any particular book is present in the institute's library. A book could be searched by its: Title, Authors name, Publisher's name
- *User login:* A registered user of LIS can login to the system by providing his employee ID and password as set by him while registering. After successful login, "Home" page for the user is shown from where he can access the different functionalities of LIS: search book, issue book, return book, reissue book. Any employee ID not registered with LIS cannot access the "Home" page – a login

failure message would be shown to him, and the login dialog would appear again. This same thing happens when any registered user types in his password wrong. However, if incorrect password has been provided for three time consecutively, the security question for the user (specified while registering) with an input box to answer it are also shown. If the user can answer the security question correctly, a new password would be sent to his email address. In case the user fails to answer the security question correctly, his LIS account would be blocked. He needs to contact with the administrator to make it active again.

- *Issue book:* Any member of LIS can issue a book against his account provided that:
 - The book is available in the library i.e. could be found by searching for it in LIS
 - No other member has currently issued the book
 - Current user has not issued the maximum number of books that can

If the above conditions are met, the book is issued to the member.

Note that this FR would remain incomplete if the "maximum number of books that can be issued to a member" is not defined. We assume that this number has been set to four for students and research scholars, and to ten for professors.

Once a book has been successfully issued, the user account is updated to reflect the same.

- *Return book:* A book is issued for a finite time, which we assume to be a period of 20 days. That is, a book once issued should be returned within the next 20 days by the corresponding member of LIS. After successful return of a book, the user account is updated to reflect the same.
- *Reissue book:* Any member who has issued a book might find that his requirement is not over by 20 days. In that case, he might choose to reissue the book, and get the permission to keep it for another 20 days. However, a member can reissue any book at most twice, after which he has to return it. Once a book has been successfully reissued, the user account is updated to reflect the information.

Identification of Non-functional Requirements

- *Performance Requirements:* This system should remain accessible 24x7; At least 50 users should be able to access the system altogether at any given time
- *Security Requirements:* This system should be accessible only within the institute LAN; The database of LIS should not store any password in plain text – a hashed value has to be stored

Practice Task

Consider the problem statement for an "Online Auction System" to be developed

New users can register to the system through an online process. By registering a user agrees to abide by different pre-defined terms and conditions as specified by the system. Any registered user can access the different features of the system authorized to him / her, after he authenticates himself through the login screen. An authenticated user can put items in the system for auction. Authenticated users can place bid for an item. Once the auction is over, the item will be sold to the user placing the maximum bid. Payments are to be made by third party payment services, which, of course, is guaranteed to be secure. The user selling the item will be responsible for its shipping. If the seller thinks he's getting a good price, he can, however, sell the item at any point of time to the maximum bidder available.

CSc 322 - Software Engineering

Lab – Spring 2017

Instructor: Arun Adiththan, email: arun.cuny@gmail.com

Use Case Modeling

March 3, 2017

Introduction

Use case diagram is a platform that can provide a common understanding for the end-users, developers and the domain experts. It is used to capture the basic functionality i.e. use cases, and the users of those available functionality, i.e. actors, from a given problem statement.

Use case diagrams belong to the category of behavioural diagram of UML diagrams. Use cases are usually identified during the early stages of the project. Use case diagrams aim to present a graphical overview of the functionality provided by the system. Thus it is highly useful for exposing requirements and planning of almost every project. It consists of a set of actions (referred to as use cases) that the concerned system can perform, one or more actors, and dependencies among them.

Actor

An actor can be defined as an object or set of objects, external to the system, which interacts with the system to get some meaningful work done. Actors could be human, devices, or even other systems. For example, consider the case where a customer *withdraws* cash from an ATM. Here, customer is a human actor.

2 types of actor classification:

- *Primary actor*: They are principal users of the system, who fulfil their goal by availing some service from the system. For example, a customer uses an ATM to withdraw cash when he needs it. A customer is the primary actor here.
- *Supporting actor*: They render some kind of service to the system. "Bank representatives", who replenishes the stock of cash, is such an example. It may be noted that replenishing stock of cash in an ATM is not the prime functionality of an ATM.

Actor Designing Consideration:

- Who/what will be interested in the system
- Who/what will want to change the data in the system
- Who/what will want to interface with the system
- Who/what will want information from the system

Usecase

A use case is simply a functionality provided by a system. Use case should ideally begin with a verb. Should not be open ended. Register, wrong. Register New User, right. Once the primary and secondary actors have been identified, we have to find out their goals i.e. what are the functionality they can obtain from the system.

Continuing with the example of the ATM, withdraw cash is a functionality that the ATM provides. Therefore, this is a use case. Other possible use cases includes, check balance, change PIN, and so on.

Use cases include both successful and unsuccessful scenarios of user interactions with the system. For example, authentication of a customer by the ATM would fail if he enters wrong PIN. In such case, an error message is displayed on the screen of the ATM.

Subject

Subject is simply the system under consideration. Use cases apply to a subject. For example, an ATM is a subject, having multiple use cases, and multiple actors interact with it. However, one should be careful of external systems interacting with the subject as actors.

Graphical Representation

An actor is represented by a stick figure and name of the actor is written below it. A use case is depicted by an ellipse and name of the use case is written inside it. The subject is shown by drawing a rectangle. Label for the system could be put inside it. Use cases are drawn inside the rectangle, and actors are drawn outside the rectangle, as shown in fig. 1.

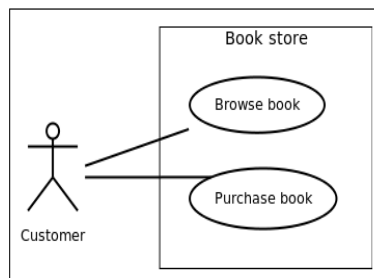


Figure 1: A use case diagram for a book store

Association between Actors and Use Cases

A use case is triggered by an actor. Actors and use cases are connected through binary associations indicating that the two communicates through message passing. An actor must be associated with at least one use case. Similarly, a given use case must be associated with at least one actor. Association among the actors are usually not shown. However, one can depict the class hierarchy among actors.

Use Case Relationships

Three types of relationships exist among use cases:

1. Include relationship
2. Extend relationship
3. Use case generalization

Include relationship

Include relationships are used to depict common behaviour that are shared by multiple use cases. This could be considered analogous to writing functions in a program in order to avoid repetition of writing the same code. Such a function would be called from different points within the program. Include relationship

is depicted by a dashed arrow with a «include» stereotype from the including use case to the included use case.

Example

For example, consider an email application. A user can send a new mail, reply to an email he has received, or forward an email. However, in each of these three cases, the user must be logged in to perform those actions. Thus, we could have a login use case, which is included by compose mail, reply, and forward email use cases. The relationship is shown in fig. 2.

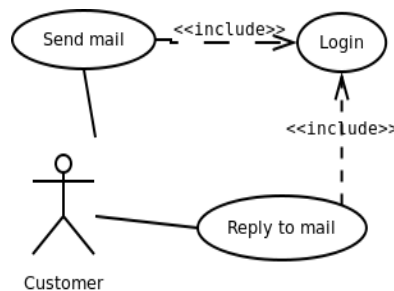


Figure 2: Include relationship between use cases

Extend Relationship

Use case extensions are used to depict any variation to an existing use case. They are used to specify the changes required when any assumption made by the existing use case becomes false. Extend relationship is depicted by a dashed arrow with a «extend» stereotype from the extending use case to the extended use case.

Example

Let's consider an online bookstore. The system allows an authenticated user to buy selected book(s). While the order is being placed, the system also allows to specify any special shipping instructions, for example, call the customer before delivery. This Shipping Instructions step is optional, and not a part of the main Place Order use case. fig. 3 depicts such relationship.

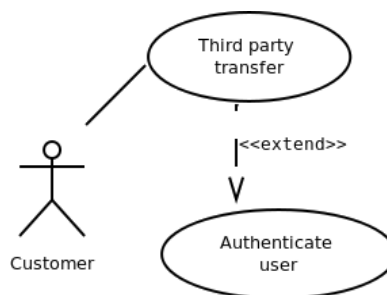


Figure 3: Extend relationship between use cases

Generalization Relationship

Generalization relationships are used to represent the inheritance between use cases. A derived use case specializes some functionality it has already inherited from the base use case. Generalization relationship

is depicted by a solid arrow from the specialized (derived) use case to the more generalized (base) use case.

Example

To illustrate this, consider a graphical application that allows users to draw polygons. We could have a use case draw polygon. Now, rectangle is a particular instance of polygon having four sides at right angles to each other. So, the use case draw rectangle inherits the properties of the use case draw polygon and overrides its drawing method. This is an example of generalization relationship. Similarly, a generalization relationship exists between draw rectangle and draw square use cases. The relationship has been illustrated in fig. 4.

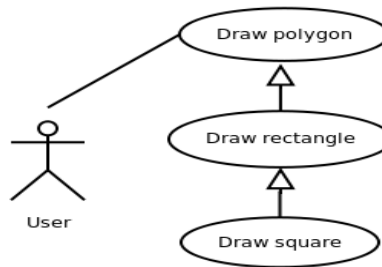


Figure 4: Generalization relationship among use cases

TASK 1

Part 1 Identify the actors and use cases for the following problem statement

An automated teller machine (ATM) lets customers to withdraw cash anytime from anywhere without requiring involvement of any banking clerk or representative. Customer must insert his ATM card into the machine and authenticate himself by typing in his personal identification number (PIN). He cannot avail any of the facilities if the PIN entered is wrong. Authenticated customers can also change their PIN. They can deposit cash to their account with the bank. Also they can transfer funds to any other account. The ATM also provides options to the user to pay electricity or phone bill. Everyday morning the stock of cash in the ATM machine is replenished by a representative from the bank. Also, if the machine stops working, then it is fixed by a maintenance guy.

Part 2 Exercise on Use Case Relationships

1. Consider a Shop Sales System. A sales officer uses the system to process/record requests of both customer order and faulty goods return. Both processes will need to identify the customer first
2. Consider an online selling/buying website. The website requires the seller and buyer to register their bank account information. A separate process will be called if the bank account is suspended/unavailable/illegitimate
3. An order management system accepts ordering of product by phone or internet. Write the generalization for this, where that base use case will be used by the order registry clerk.

Part 3 Mail Order System

Identify Actors and Use Cases for the following problem and draw a use case diagram using ArgoUML tool with appropriate relationships

In order to improve the operational efficiency of a mail order company, the chief executive officer is interested in computerizing the company's business process. The major business activities of the company can be briefly described as follows:

A customer registers as a member by filling in the membership form and mailing it to the company. A member who has not been active (no transactions made) for a period of one year will be removed from the membership list and he/she needs to re-apply for the reinstatement of the lapsed membership.

A member should inform the company of any changes of personal details such as home address, telephone numbers, etc. A member can place an order by filling out a sales order form and faxing it to the company or by phoning the Customer Service Assistant with the order details.

The Customer Service Assistant first checks for the validity of membership and enters the sales order information into the system.

The Order Processing Clerk checks the availability of the ordered items and holds them for the order. When all the ordered items are available, he/she will schedule their delivery.

The Inventory Control Clerk controls and maintains an appropriate level of stock and is also responsible for acquiring new items.

If there is a problem with an order, members will phone the Customer Service Assistant. The Customer Service Assistant will take appropriate action to follow up the sales order.

Members may return defective goods within 30 days and get their money back.

The system will record the name of the staff member who has initialized an updated transaction to the system.

CSc 322 - Software Engineering

Lab – Spring 2017

Instructor: Arun Adiththan, email: arun.cuny@gmail.com

UML Class and Sequence diagrams

March 24, 2017

Classes are the structural units in object oriented system design approach, so it is essential to know all the relationships that exist between the classes, in a system. All objects in a system are also interacting to each other by means of passing messages from one object to another. Sequence diagram shows these interactions with time ordering of the messages.

Class diagram

Elements in class diagram

Class diagram contains the system classes with its data members, operations and relationships between classes

Class

A set of objects containing similar data members and member functions is described by a class. In UML syntax, class is identified by solid outline rectangle with three compartments which contain

- *Class name*: A class is uniquely identified in a system by its name. A textual string [2] is taken as class name. It lies in the first compartment in class rectangle
- *Attributes*: Property shared by all instances of a class. It lies in the second compartment in class rectangle
- *Operations*: An execution of an action can be performed for any object of a class. It lies in the last compartment in class rectangle

Example: To build a structural model for an Educational Organization, 'Course' can be treated as a class which contains attributes 'courseName' & 'courseID' with the operations 'addCourse()' & 'removeCourse()' allowed to be performed for any object to that class

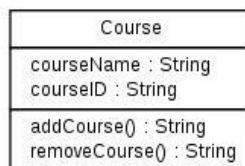


Figure 1: Simple Class Diagram for a Course

- *Generalization/Specialization*: It describes how one class is derived from another class. Derived class inherits the properties of its parent class

Example: Geometric_Shapes is the class that describes how many sides a particular shape has. Triangle, Quadrilateral and Pentagon are the classes that inherit the property of the Geometric_Shapes class. So the relations among these classes are generalization. Now Equilateral_Triangle, Isosceles_Triangle and Scalene_Triangle, all these three classes inherit the properties of Triangle class as each one of them has three sides. So, these are specialization of Triangle class.

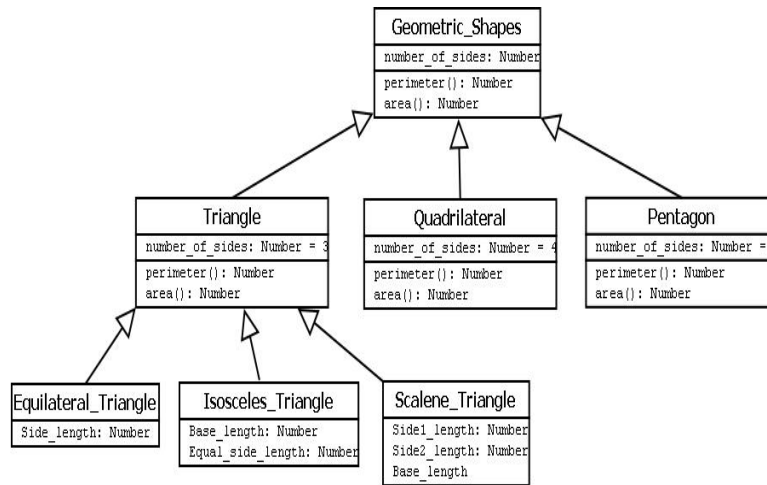


Figure 2: Generalization/Specialization: Shapes

Relationships

Existing relationships in a system describe legitimate connections between the classes in that system

- *Association*

It is an instance level relationship that allows exchanging messages among the objects of both ends of association. A simple straight line connecting two class boxes represent an association. We can give a name to association and also at the both end we may indicate role names and multiplicity of the adjacent classes. Association may be uni-directional.

Example: In structure model for a system of an organization an employee (instance of 'Employee' class) is always assigned to a particular department (instance of 'Department' class) and the association can be shown by a line connecting the respective classes

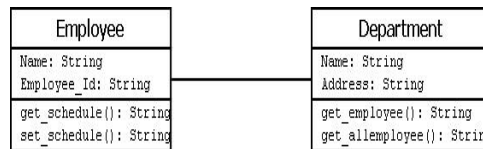


Figure 3: Relationship (Association): Employee-Department

- *Aggregation*

It is a special form of association which describes a part-whole relationship between a pair of classes. It means, in a relationship, when a class holds some instances of related class, then that relationship can be designed as an aggregation

Example: For a supermarket in a city, each branch runs some of the departments they have. So, the relation among the classes 'Branch' and 'Department' can be designed as an aggregation. In UML, it can be shown as in the fig. below

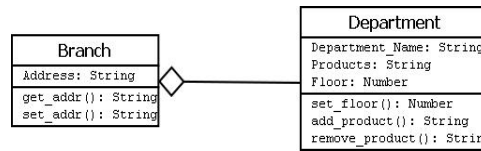


Figure 4: Relationship (Aggregation): Branch-Department

- *Composition*

It is a strong form of aggregation which describes that whole is completely owns its part. Life cycle of the part depends on the whole

Example: Let consider a shopping mall has several branches in different locations in a city. The existence of branches completely depends on the shopping mall as if it is not exist any branch of it will no longer exists in the city. This relation can be described as composition and can be shown as below

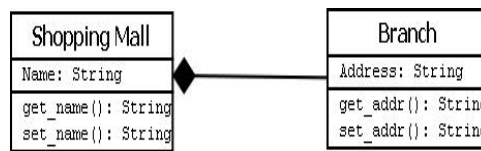


Figure 5: Relationship (Composition): Branch-Shopping Mall

- *Multiplicity*

It describes how many numbers of instances of one class is related to the number of instances of another class in an association.

Notations for different types of multiplicity

- # Single instance – 1
- # Zero or one instance – 0..1
- # Zero or more instance – 0..*
- # One or more instance – 1..*
- # Specific range (say, 2 to 6) – 2..6

Examples: One vehicle may have two or more wheels

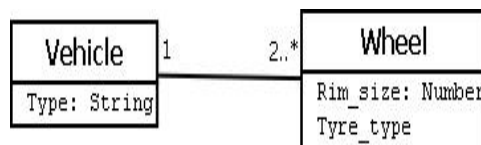


Figure 6: Multiplicity

Sequence diagram

It represents the behavioral aspects of a system. Sequence diagram shows the interactions between the objects by means of passing messages from one object to another with respect to time in a system

Elements in sequence diagram

Sequence diagram contains the objects of a system and their life-line bar and the messages passing between them

Object

Objects appear at the top portion of sequence diagram. Object is shown in a rectangle box. Name of object precedes a colon ':' and the class name, from which the object is instantiated. The whole string is underlined and appears in a rectangle box. Also, we may use only class name or only instance name.

Objects which are created at the time of execution of use case and are involved in message passing , are appear in diagram, at the point of their creation

Life-line bar

A down-ward vertical line from object-box is shown as the life-line of the object. A rectangle bar on life-line indicates that it is active at that point of time

Messages

Messages are shown as an arrow from the life-line of sender object to the life-line of receiver object and labelled with the message name. Chronological order of the messages passing throughout the objects life-line show the sequence in which they occur. There may exist some different types of messages

- *Synchronous messages*: Receiver start processing the message after receiving it and sender needs to wait until it is made. A straight arrow with close and fill arrow-head from sender life-line bar to receiver end, represent a synchronous message
- *Asynchronous messages*: For asynchronous message sender needs not to wait for the receiver to process the message. A function call that creates thread can be represented as an asynchronous message in sequence diagram. A straight arrow with open arrow-head from sender life-line bar to receiver end, represent an asynchronous message
- *Return message*: For a function call when we need to return a value to the object, from which it was called, then we use return message. But, it is optional, and we are using it when we are going to model our system in much detail. A dashed arrow with open arrow-head from sender life-line bar to receiver end, represent that message
- *Response message*: One object can send a message to self. We use this message when we need to show the interaction between the same object




Message Type	Notation
Synchronous message	
Asynchronous message	
Response message	

Figure 7: Sequence Diagram Notations

Practice Tasks

Part 1 Represent the three-way handshaking mechanism of TCP with a sequence diagram

Part 2 Draw class diagram for University courses application described below

For each course maintain a list of the students on that course and the lecturer who has been assigned to teach that course. Allow options such as adding and removing of students from the course, assigning a teacher, getting a list of the currently assigned students, and the currently assigned teacher for each course. Teachers are modelled as Lecturer objects. A lecturer may teach more than one course. Each Lecturer object also maintains a list of the Courses that it teaches. Similarly, a course is attended by zero or more students, and a student may attend multiple courses.

Part 3 Class and Sequence Diagram of a Web Browser

A web browser is a software that helps to access a resource (web page) available on the World Wide Web and identified by a URL. Whenever a user types in the URL of a web page in the browser's address bar and clicks the "Go" button, the browser sends a HTTP request to the concerned web server. If the requested resource is available and accessible, the web server sends back a HTTP response to the requesting web browser. In case of any error, a HTTP response is sent indicating the error.

When the web browser receives a HTTP response, it displays the web page to the user. In very simple terms a web browser can be thought of consisting the following sub-components: rendering engine, and browser control.

Once a HTTP response has been obtained from the server, the rendering engine decides the layout of the contents and actually displays the requested page. This is done keeping in mind the different HTML elements that are present in the page, and corresponding CSS rules, if any.

The browser control provides facilities like navigating across pages (by following hyperlinks), reload a page, and handles other events related to the window display, for example, resizing the browser window.

Instruction: Represent classes with their state and behaviour. Identify and represent association, aggregation, composition, and inheritance of classes as needed. Identify sequence of activities and draw the sequence diagrams.

Part 4 Determine a useful application of your own choice, and draw class and sequence diagram for it. Make sure you write the problem description clearly. (Note: Don't try to take and solve a trivial problem. It should be at least as substantial as the one in Part 3)

CSc 322 - Software Engineering

Lab – Spring 2017

Instructor: Arun Adiththan, email: arun.cuny@gmail.com

Entity Relationship Diagram

April 4, 2017

Entity, Attribute, and Relationship

- **Entity:**

- Entity class (entity set) is a structural description of things that share common attributes
- Entity instance is the occurrence of a particular entity

- **Attribute:**

- Describes an entity class
- All entity instances of a given entity class have the same attributes, but vary in the values of those attributes
- **Attribute Types:**
 - * *Composite attribute:* An attribute that can be further divided into more attributes. Example: Name, Address, etc
 - * *Multi-Value Attribute:* An attribute that allow multiple values. Example: skills, phone numbers, etc.
 - * *Derived attribute:* Attributes that can be calculated (derived) from other attributes. Example: age, total, interest, due date, etc.

- **Identifier:**

- Identifies an entity instance
- The value of the identifier attribute is unique for each entity instance

- **Relationship**

- Relationship describes how entities are related
- Relationship features
 - * *Cardinality:* Entity instance's participation count
 - * *Degree of Relationship:* How many entities are involved in a relationship?
- Cardinality
 - * Describes how many entity instance can be in the relationship
 - * Maximum cardinality (type of relationship): Describes the maximum number of entity instances that participate in a relationship – One-to-one, One-to-many, Many-to-many.
 - * Minimum cardinality: Describes the minimum number of entity instances that must participate in a relationship

Notation

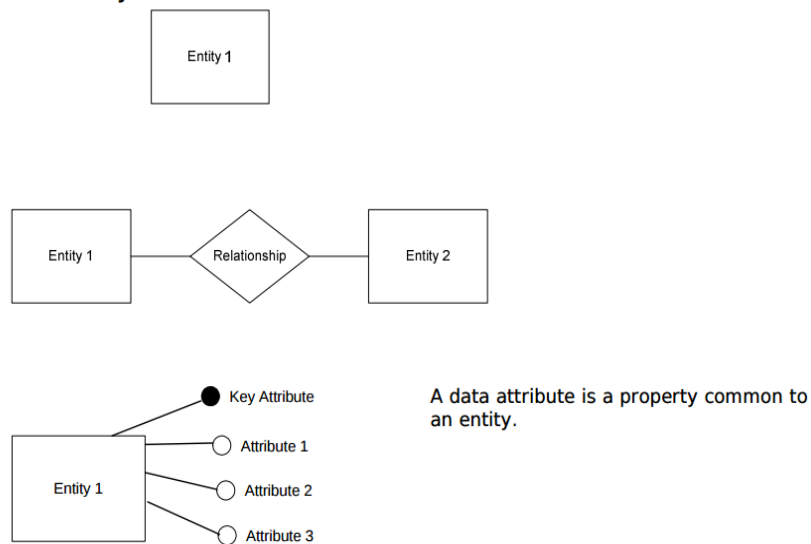


Figure 1: Graphical notations

Steps to create an ER Diagram

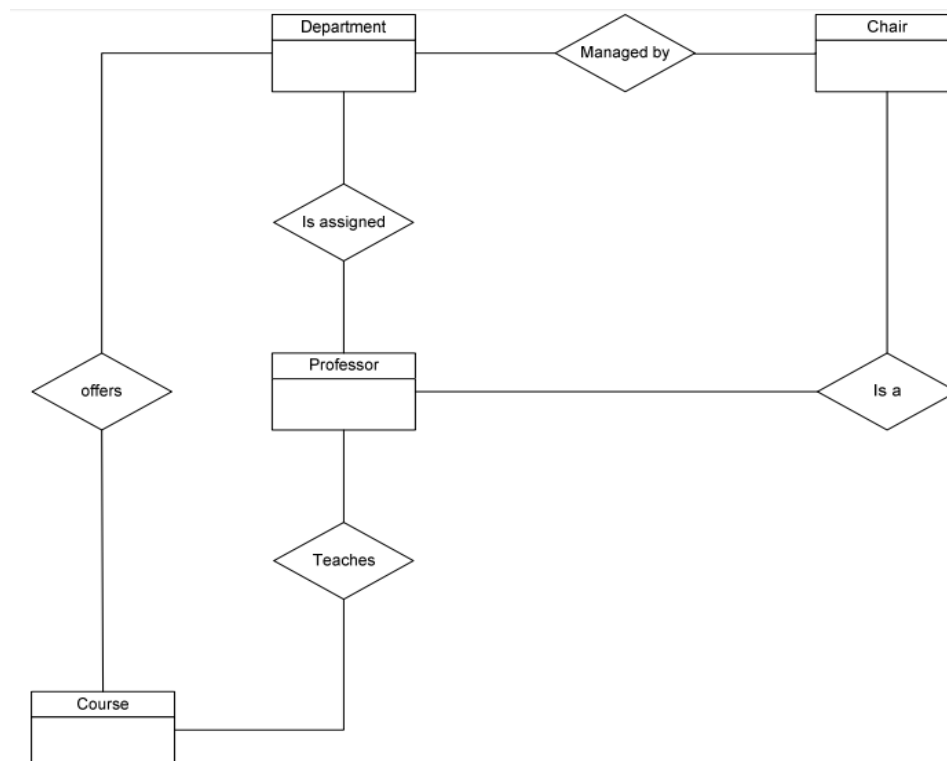
1. **Identify Entities:** Identify the entities. These are typically the nouns and noun-phrases in the descriptive data produced in your analysis. Do not include entities that are irrelevant to your domain.
2. **Find Relationships:** Discover the semantic relationships between entities. These are usually the verbs that connect the nouns. Not all relationships are this blatant, you may have to discover some on your own. The easiest way to see all possible relationships is to build a table with the entities across the columns and down the rows, and fill in those cells where a relationship exists between entities.
3. **Draw Rough ERD:** Draw the entities and relationships that you have discovered.
4. **Fill in Cardinality:** Determine the cardinality of the relationships. You may want to decide on cardinality when you are creating a relationship table.
5. **Define Primary Keys:** Identify attribute(s) that uniquely identify each occurrence of that entity.
6. **Draw Key-Based ERD:** Now add them (the primary key attributes) to your ERD. Revise your diagram to eliminate many-to-many relationships, and tag all foreign keys .
7. **Identify Attributes:** Identify all entity characteristics relevant to the domain being analyzed.
8. **Map Attributes:** Determine which to entity each characteristic belongs. Do not duplicate attributes across entities. If necessary, contain them in a new, related, entity.
9. **Draw fully attributed ERD:** Now add these attributes. The diagram may need to be modified to accommodate necessary new entities.

Example The University of Toronto has several departments. Each department is managed by a chair, and at least one professor. Professors must be assigned to one, but possibly more departments. At least one professor teaches each course, but a professor may be on sabbatical and not teach any course. Each course may be taught more than once by different professors. We know of the department name, the professor name, the professor employee id, the course names, the course schedule, the term/year that the course is taught, the departments the professor is assigned to, the department that offers the course.

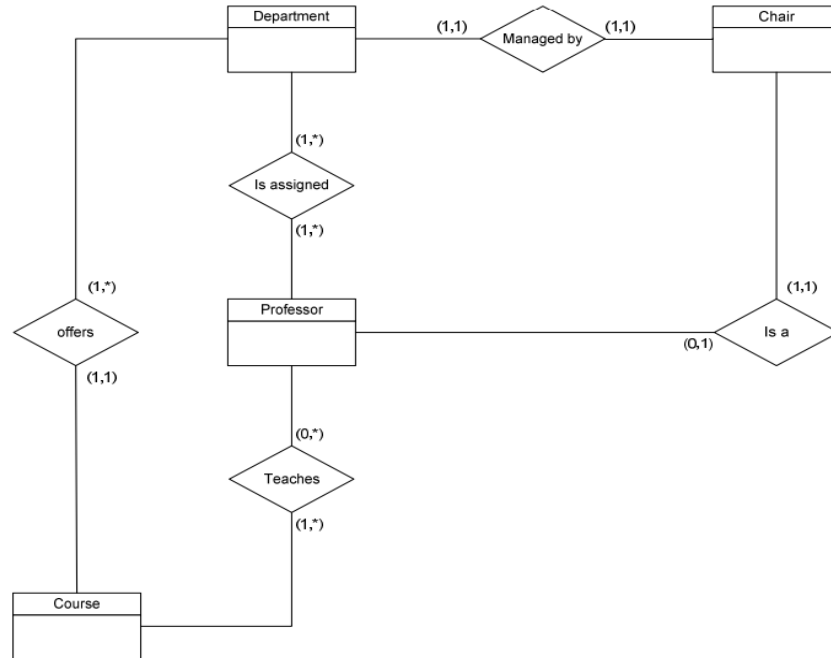
1. **Identify Entities** Identify the entities. These are typically the nouns and noun-phrases in the descriptive data produced in your analysis. Do not include entities that are irrelevant to your domain. The entity candidates are departments, chair, professor, course, and course section. Since there is only one instance of the University of Toronto, we exclude it from our consideration.
2. **Find Relationships** Discover the semantic relationships between entities. These are usually the verbs that connect the nouns. Not all relationships are this blatant, you may have to discover some on your own. The easiest way to see all possible relationships is to build a table with the entities across the columns and down the rows, and fill in those cells where a relationship exists between entities.

	department	chair	professor	course
department		managed by	is assigned	offers
chair	manages		is a	
professor	assigned to			teaches
course	offered by		taught by	

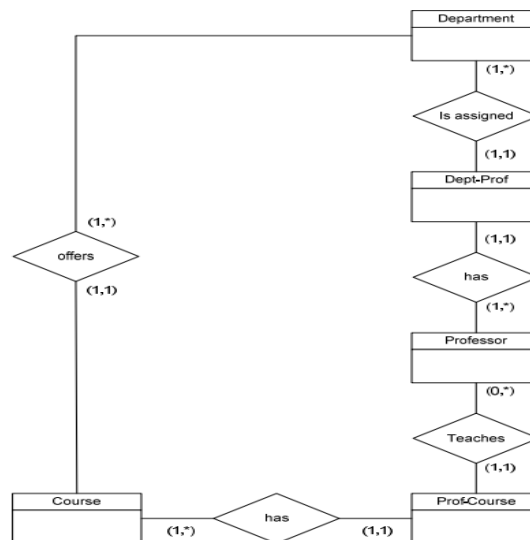
3. Draw Rough ERD



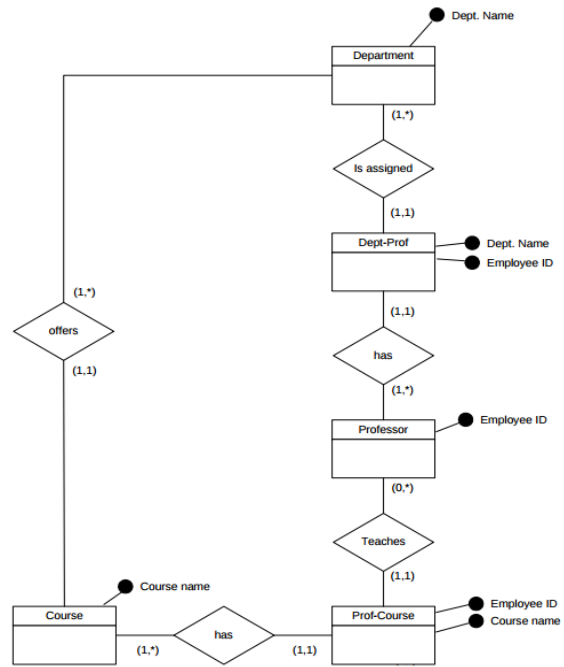
4. Fill in Cardinality



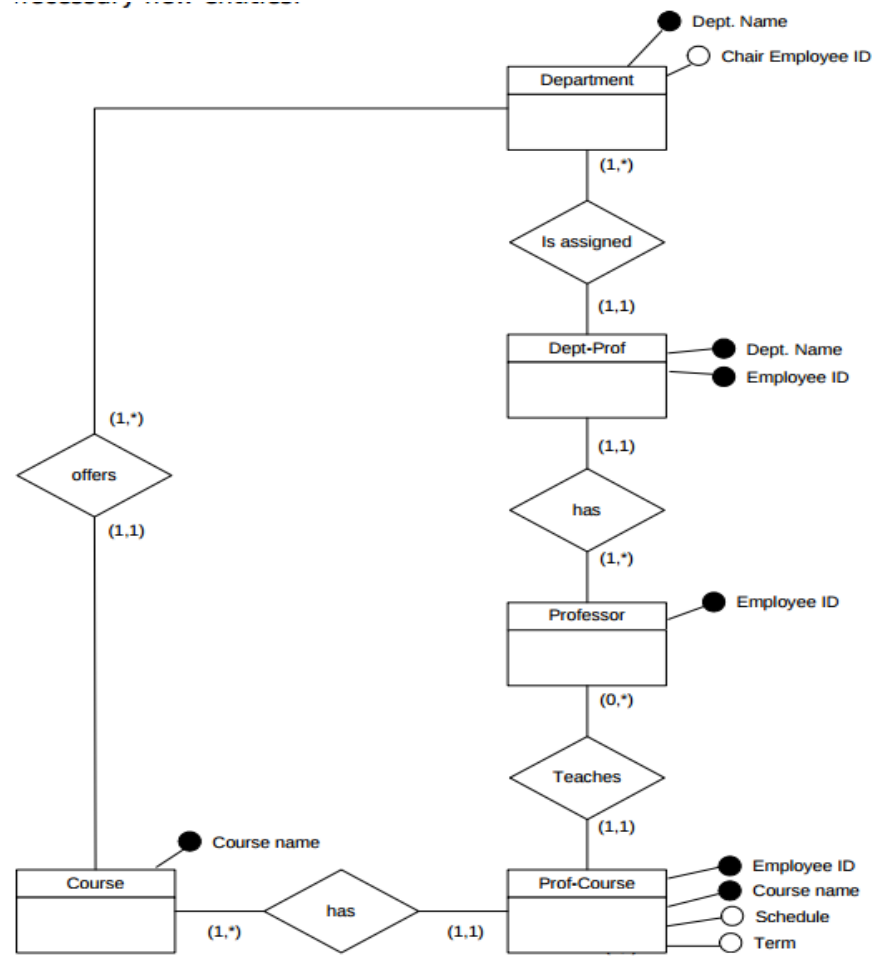
Here we must eliminate many-to-many relationships, and collapse one-to-one relationships where it makes sense. For example, the chair, without any behaviours, is really just an attribute of a department. So we can remove it as an entity and later add it as an attribute.



5. Draw Key-Based ERD Now add them (the primary key attributes) to your ERD. Revise your diagram to eliminate many-to-many relationships, and tag all foreign keys .



6. **Identify Attributes** Identify all entity characteristics relevant to the domain being analyzed. Excluding those keys already identified: Schedule, Term, Professor name, Department Chair (which is an employee ID, a foreign key to Professor)
7. **Map Attributes** Determine which to entity each characteristic belongs. Do not duplicate attributes across entities. If necessary, contain them in a new, related, entity. Schedule Prof-Course, Term Prof-Course, Chair Department
8. **Draw fully attributed ERD** Now add these attributes. The diagram may need to be modified to accommodate necessary new entities.



CSc 322 - Software Engineering

Lab – Spring 2017

Instructor: Arun Adiththan, email: arun.cuny@gmail.com

Collaboration Diagram

April 21, 2017

Interaction diagrams are models that describe how a group of objects collaborate in some behavior - typically a single use-case. The diagrams show a number of example objects and the messages that are passed between these objects within the use-case. Interaction diagrams should be used when you want to look at the behavior of several objects within a single use case. They are good at showing the collaborations between the objects, they are not so good at precise definition of the behavior.

Notation

Diagrams represent a collaboration and interaction. Collaboration captures a set of objects and their interactions in a specific contexts. Whereas, an interaction is a set of messages exchanged in collaboration to produce a desired result. The objects are typically represented as follows:

- rectangles containing object signature
- object signature: object name: object class
 - object name is optional and starts with a lowercase letter
 - class name is mandatory and starts with a uppercase letter
 - objects are connected by lines
- Messages are labeled like function calls in languages like C. Message names are followed by brackets and can have parameters and return values. Messages are typically written on top of an arrow that show the direction of message flow. Simple collaboration diagrams simply number the messages in sequence. More complex schemes use a decimal numbering approach to indicate if messages are sent as part of the implementation of another message. In addition a letter can be used to show concurrent threads.
- One can specify conditions that need to be satisfied to successfully execute the message as follows:
seq_no: [condition] Message.

Examples

Order Processing: In this behavior the order entry window sends a prepare message to an order. The order then sends prepare to each order line on the order. The order line first checks the stock item, and if the check returns true it removes stock from the stock item. If stock item falls below the reorder level it requests a new delivery. (see Fig. 1)

Bank: Withdrawal transaction (see Fig. 2)

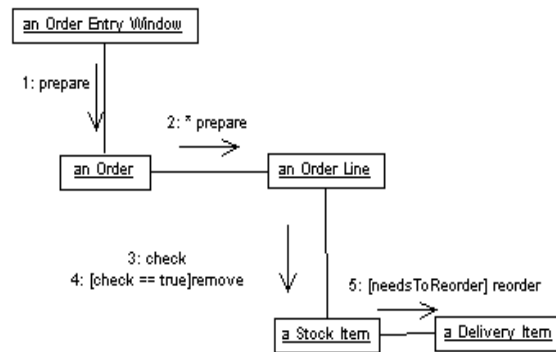


Figure 1: Order Processing

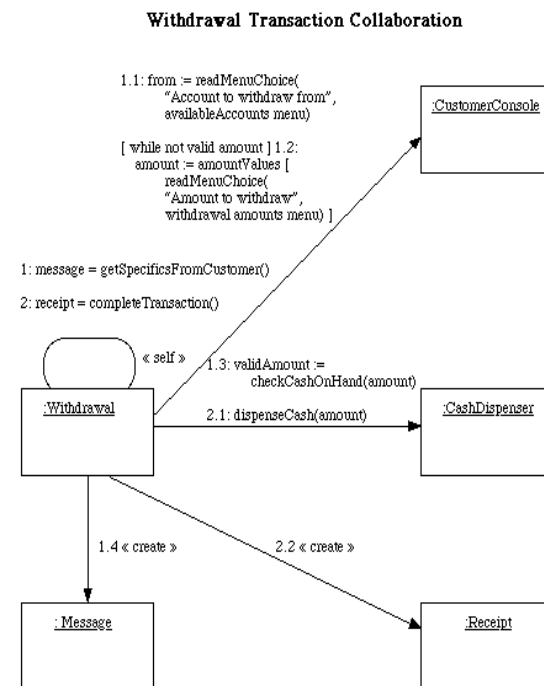


Figure 2: Withdrawal Transaction Collaboration