Requirement Modeling Strategies

- The requirements model has many different dimensions.about flow oriented models, behavioral models, and the special requirements analysis considerations that come into play when WebApps are developed

- One view of requirements modeling, called ***structured analysis***, considers **data and the processes that transform the data as separate entities**.

- **Data objects are modeled in a way that defines their attributes and relationships**. Processes that manipulate data objects are modeled in a manner that shows how they transform data as data objects flow through the system.

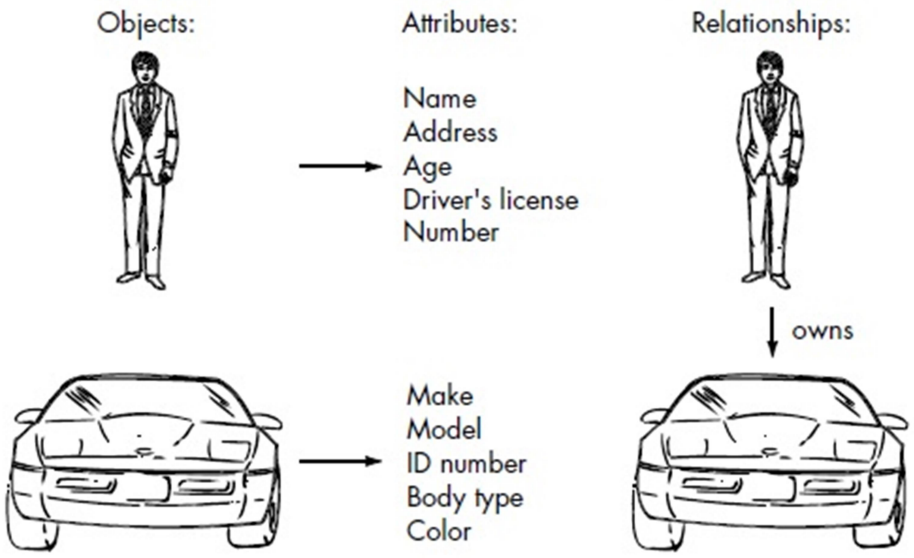
- A second approach to analysis modeled, called **object-oriented analysis**, focuses on the definition of classes and the manner in which **they collaborate with one another to effect customer requirements**.

Data Modeling

- Very useful in data processing application.
- Helpful in in identifying the primary data objects to be processed by the system
- Helpful in the composition of each data object and their attributes , and their relationships between

Data Objects, Attributes, and Relationships

- The data model consists of three interrelated pieces of information: the data object, the and the relationships objects to one another



Data objects.

- A data object can be an external entity, a thing, an occurrence or event (e.g., an alarm), a role (e.g., salesperson), an organizational unit (e.g., accounting department), a place (e.g., a warehouse), or a structure (e.g., a file).

- For example,a person or a car

- Data objects (represented in bold) are related to one another. For example, person can own car, where the relationship own

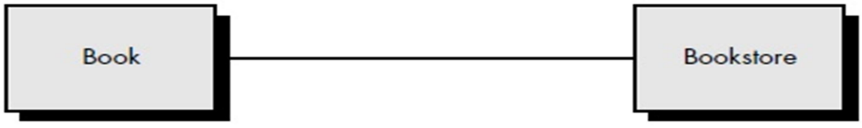
Attributes.

- Attributes define the properties of a data object and take on one of three

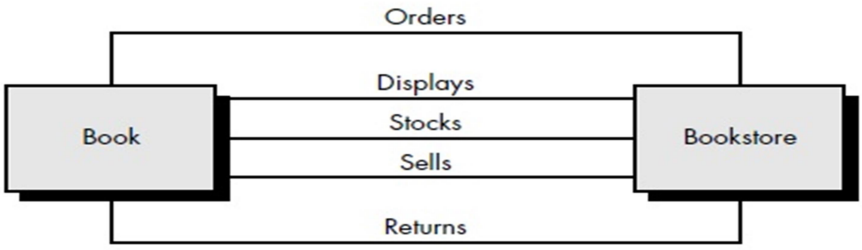
- They can be used to (1) name an instance of the data object,
  (2) describe the instance, or
  (3) make reference to another instance in another table.
- Example:The attributes for car might
  ID number,
  body type and color,
  interior code,
  transmission type , would have to be added more

Relationships.

- Data objects are connected to one another in different ways. Consider
- two data objects, book and bookstore. These objects can be represented using the simple notation illustrated in Figure
- A connection is established between book and bookstore because the two objects are related. But what are the relationships?
  • A bookstore orders books.
  • A bookstore displays books.
  • A bookstore stocks books.
  • A bookstore sells books.
  • A bookstore returns books.



(a) A basic connection between objects



(b) Relationships between objects

Cardinality

- how many occurrences of object X are related to how many occurrences of objectY. This leads to a data modeling concept called cardinality.

- Cardinality is the specification of the number of occurrences of one object that can be related to the number of occurrences of another object.

- Cardinality is usually expressed as **simply 'one' or 'many".**

    - One-to-one (l:l)—

An occurrence of object'A' can relate to one and only one occurrence of object 'B,' and an occurrence of 'B' can relate to only one occurrence of 'A.'

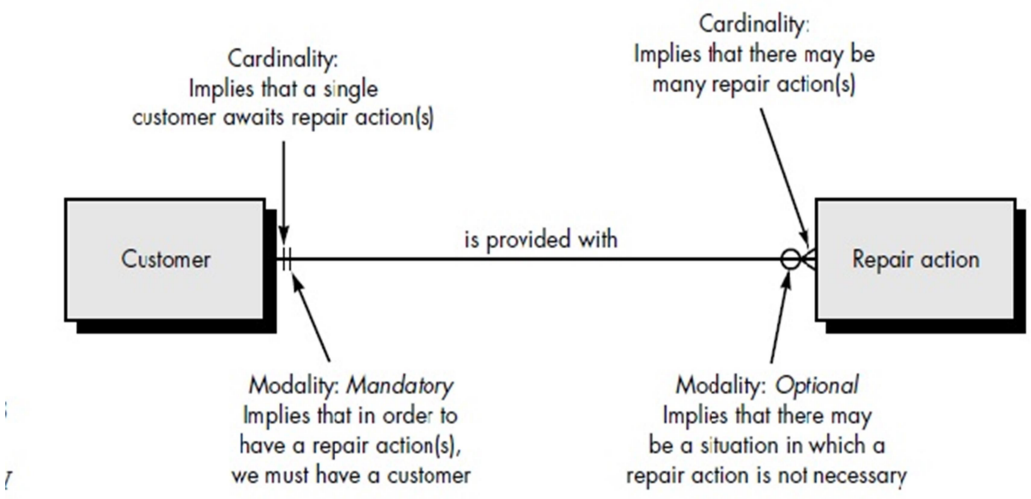
- Example, a husband have a wife

- One-to-many (l:N)—One occurrence of object 'A' can relate to one or many occurrences of 'B,' but an occurrence of 'B' can relate to only one occurrence of 'A.'
    - For example, a mother can have many children, but a child can have only one mother.

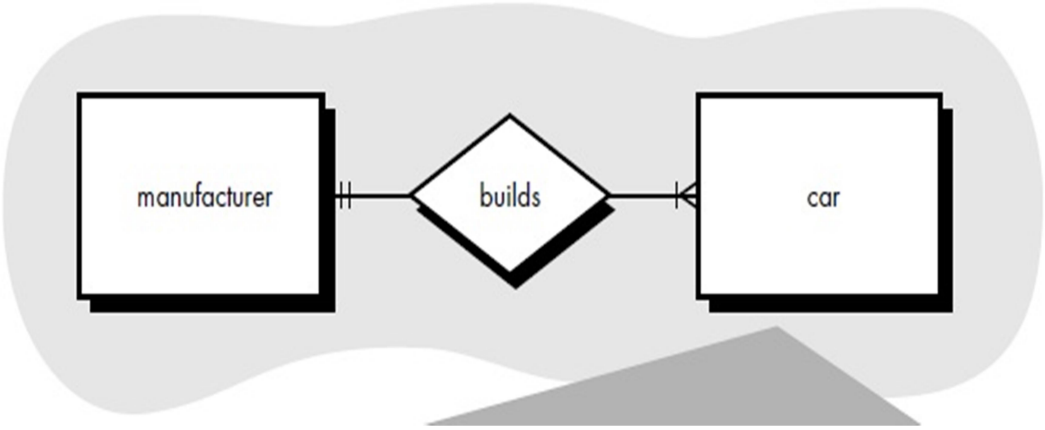
- Many-to-many (M:N)—An occurrence of 'A' can relate to one or more occurrences of 'B,' while an occurrence of 'B' can relate to one or more occurrences of 'A.'
    - For example, an uncle can have many nephews, while a nephew can have many uncles.

Modality.

- The modality of a relationship is 0 if there is no explicit need for the relationship to occur or the relationship is **optional.**
- The modality is 1 if an occurrence of the relationship is **mandatory.**
- To illustrate, consider software that is used by a local telephone company to process requests for field service. A customer indicates that there is a problem. If the problem is diagnosed as relatively simple, a single repair action occurs. However, if the problem is complex, multiple repair actions may be required.



Ex: person – hobby (perform, Have)
Student – book ,
Student- course
CUSTOMER- ORDER
Godown –stores- item
Customer –purchase- item
Prepare ERD for Railway seat booking system, showing cardinality and modality



Cardinality:
1 manufacturer can builds many cars
1 car can be build by 1 manufacturer

Modality:
Builds-> mandatory

Entity/Relationship Diagrams

- ERD comprises : data objects, attributes, relationships, and various type indicators.

- The primary purpose of the ERD is to represent data objects and their relationships.

- Data objects are represented by a labeled rectangle.
- Relationships are indicated with a labeled line connecting objects.
- ERD, the connecting line contains a diamond that is labeled with the relationship. Connections between data objects and relationships are established using a variety of special symbols that indicate cardinality and modality

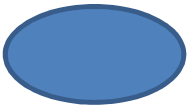
FLOW-ORIENTED MODELING

Data Flow Diagram DFD

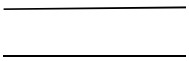
- The DFD takes an **input-process-output** view of a system.
- That is, data objectsflow into the software, are transformed by processing elements, and resultant data objects flow out of the software.
- 2 horizontal parallel lines- Data objects are represented by labeled arrows, and
  transformations are represented by circles (also called bubbles). The DFD is presented
- in a hierarchical fashion. That is, the first data flow model (sometimes called
- a level 0 DFD or context diagram) represents the system as a whole. Subsequent data
- flow diagrams refine the context diagram, providing increasing detail with each
- subsequent level

Shows External Entities

Shows flow of data into system



Shows, process that transform data i/p to data o/p

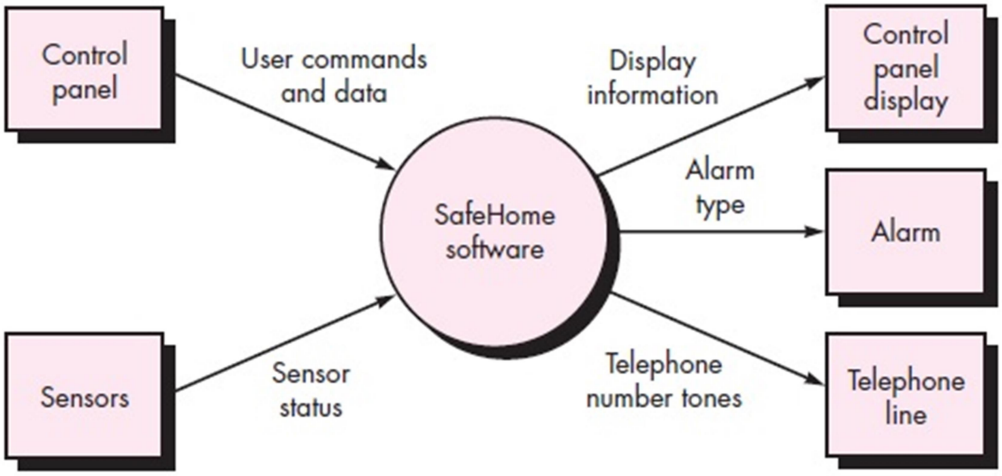


Shows Data Repository/ database

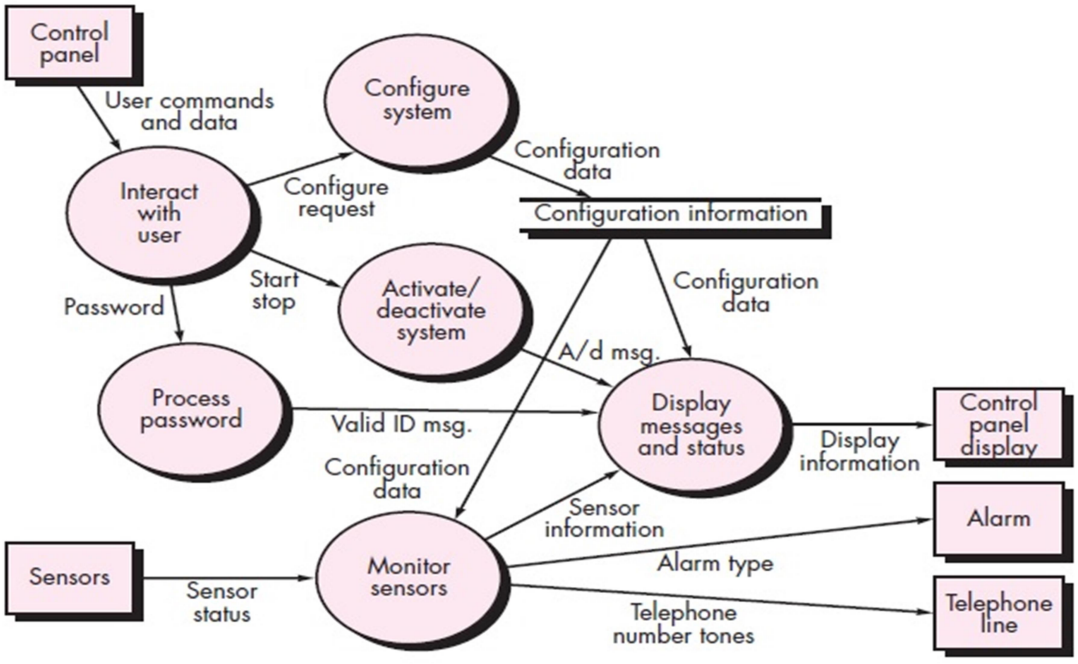
- Level 0: Shows Context level process
  level 0 data flow diagram shows-single bubble; primary input and output should be carefully noted;
- Level 1: Single process expanded with multiple processeses

- Level 2: The processes represented at DFD level 1 can be further refined into lower levels

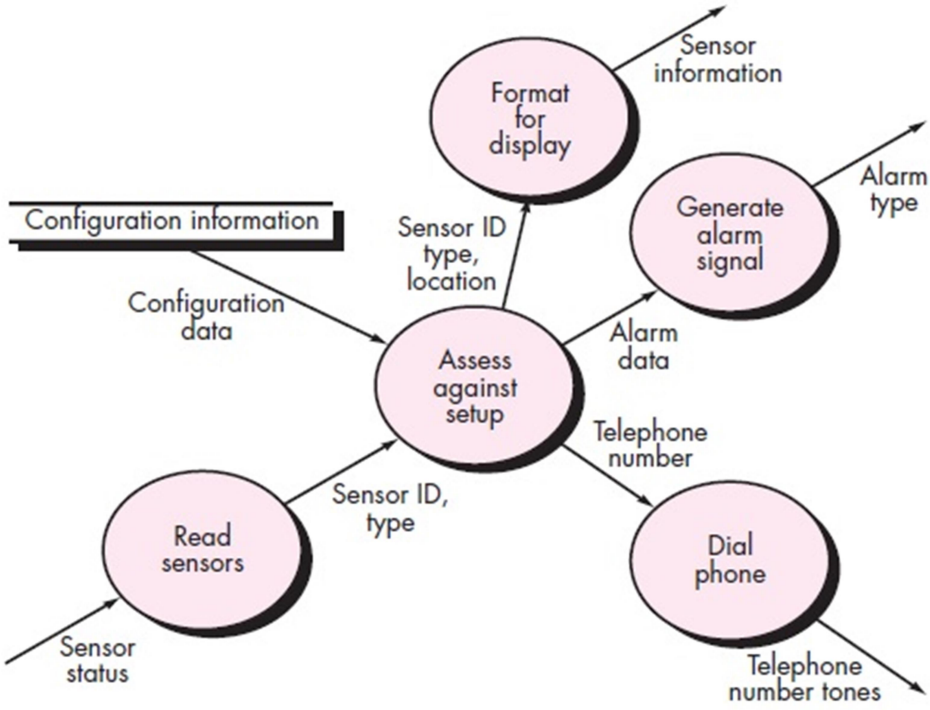
Level 0



Level 01



Level 02



Examples:1.Draw a DFD forATM system

2.Draw a DFD for Online word to pdf converter system

3.Draw a DFD for ONLINE SHOPPING SYSTEM

4.Draw DFD for Water wending machine

Control Flow Model

For some types of applications, the data model and the data flow diagram are all that is necessary to obtain meaningful insight into software requirements.

Such applications require the use of control flow modeling in addition to **data flow modeling**.

An event or control item is implemented as a Boolean value.

The following guidelines are for select potential candidate events:

· List all sensors that are “read” by the software.

· List all interrupt conditions.

· List all “switches” that are actuated by an operator.

· List all data conditions.

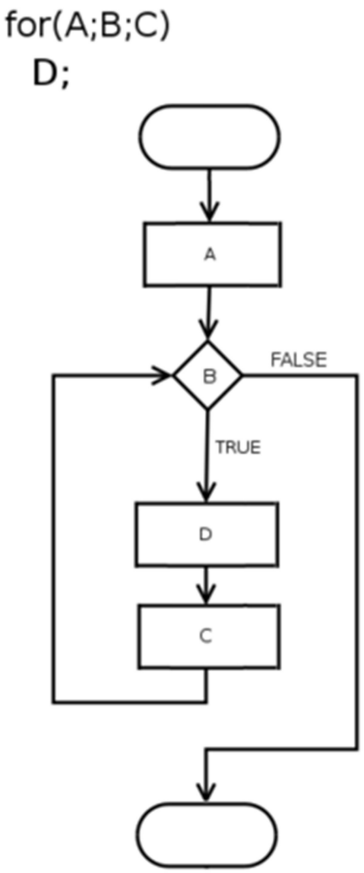
· Recalling the noun/verb parse that was applied to the processing narrative, review all “control items” as possible control specification inputs/outputs.

· Describe the behavior of a system by identifying its states, identify how each state is reached, and define the transitions between states.

· Focus on possible omissions—a very common error in specifying control; for example, ask: “Is there any other way I can get to this state or exit from it?”

The kinds of control flow statements supported by different languages vary, but can be categorized by their effect:

- Continuation at a different statement (unconditional branch or jump)
- Executing a set of statements only if some condition is met (choice - i.e., conditional branch)
- Executing a set of statements zero or more times, until some condition is met (i.e., loop - the same as conditional branch)
- Executing a set of distant statements, after which the flow of control usually returns (subroutines, coroutines, and continuations)
- Stopping the program, preventing any further execution (unconditional halt)



The Control Specification

A **control specification** (CSPEC) represents the **behavior of the system** in two different ways.

The CSPEC contains a **state diagram** that is a sequential specification of behavior. It can also contain a **program activation table** a **combinatorial** specification of behavior.

A somewhat different mode of behavioral representation is the process activation table. The PAT represents information contained in the state diagram in the context of processes, not states.

That is, the table indicates which processes (bubbles) in the flow model will be invoked when an event occurs.

The PAT can be used as a guide for a designer who must build an executive that controls the processes represented at this level.

The CSPEC describes the behavior of the system, but it gives us no information about the inner working of the processes that are activated as a result of this behavior.

The Process Specification

The **process specification** (PSPEC) is used to describe all flow model processes that appear at the **final level of refinement**.

The content of the process specification can include narrative text, a program design language (PDL) description of the process algorithm, mathematical equations, tables, or UML activity diagrams.

By providing a PSPEC to accompany each bubble in the flow model, you can create a “mini-spec” that serves as a guide for design of the software component that will implement the bubble.

- akare, devid
- 
  Baghel, Shruti
- 
  Barhate, Shamal
- 
  Barhate, Shamal
- 
  Charde, Indrajeet
- 
  Dhengale, Rajani
- 
  Gajbhiye, Gayatri
- 
  Gupta, Tarun
- 
  Jagdale, Karan
- 
  Mahalle, Piyush
- 
  Manapure, Ashish
- 
  Pakidde, Anuja
- 
  Rahman, Saif
- 
  Shirbhate, Hemant
- 
  Syed, Noor

Creating a Behavioral Model

- To make a transition to the **dynamic behavior** of the system.represent the behavior of the system as a function of specific events and time.

- The ***behavioral model*** **indicates how software will respond to external events or external stimuli**. To create the model, should perform the following steps:

1. **Evaluate all use cases** to fully understand the sequence of interaction Within the system.

2. **Identify events** that drive the interaction sequence and understand how These events relate to specific objects.

3. **Create a sequence for each use case**.

4. **Build a state diagram** for the system.

5. **Review the behavioral model to verify accuracy and consistency**.
Each of these steps is discussed in the sections that follow.

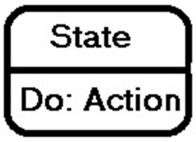
Identifying Events with the Use Case

- The use case represents a sequence of activities that involves
- actors and the system. In general, an event occurs whenever the system and an actor exchange information.
- A use case is examined for points of information exchange.
- The underlined portions of the use case scenario indicate events.
- An actor should be identified for each event; the information that is exchanged should be noted, and any conditions or constraints should be listed.

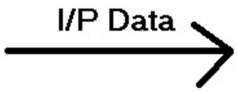
State Diagram

 - **Start**

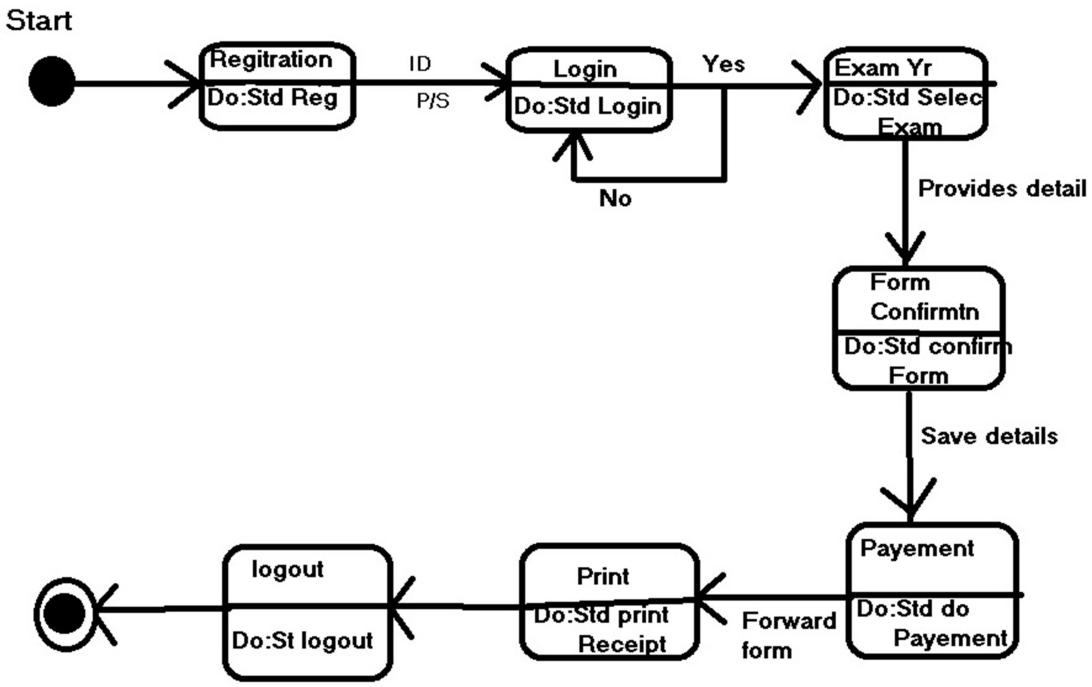
-Stop



State



Input data.



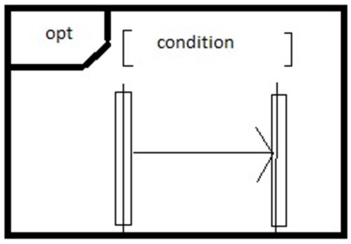
Ex: Prepare state diagram for

1. **New email account opening**
2. **Computer hardware assembling**

Sequence Diagram (Notations)

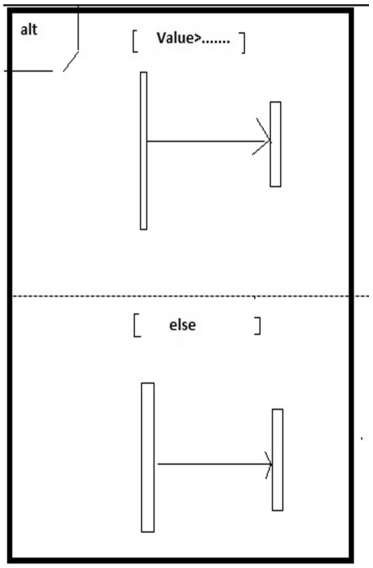
Object/Actor

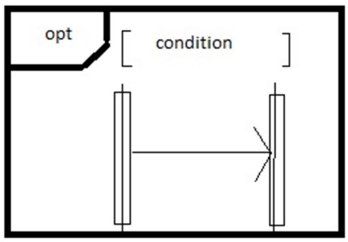
Active on_lifeline



If

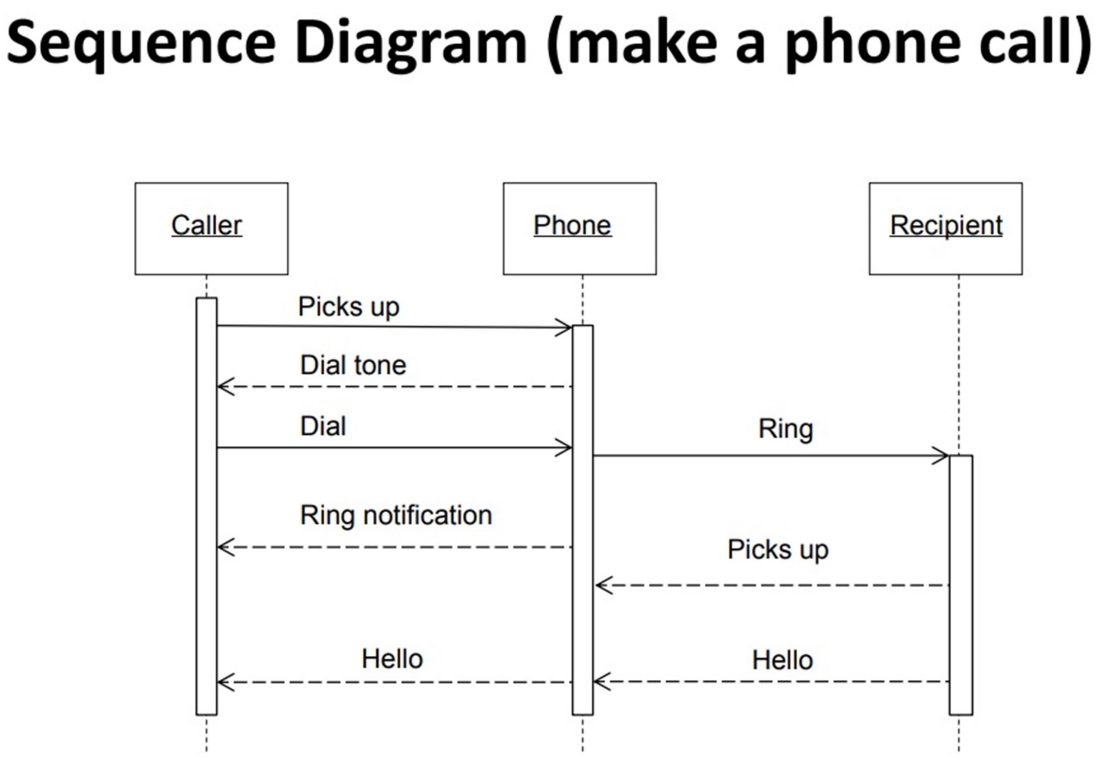
If else





Loop

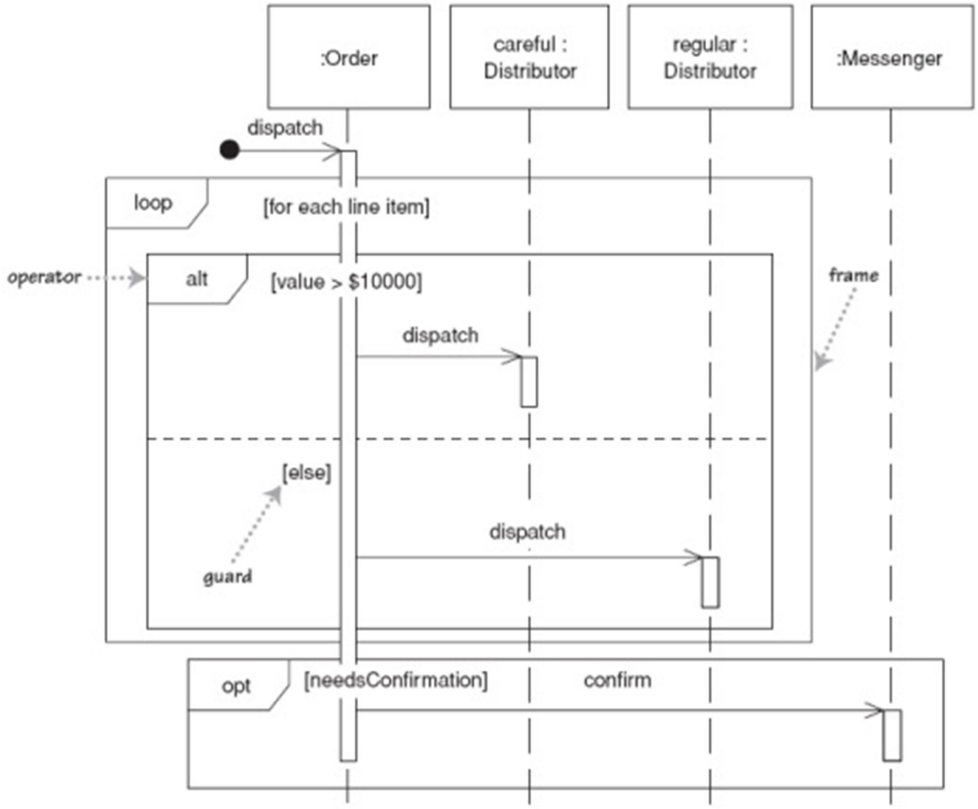
Example:



H/w

1. Client Server

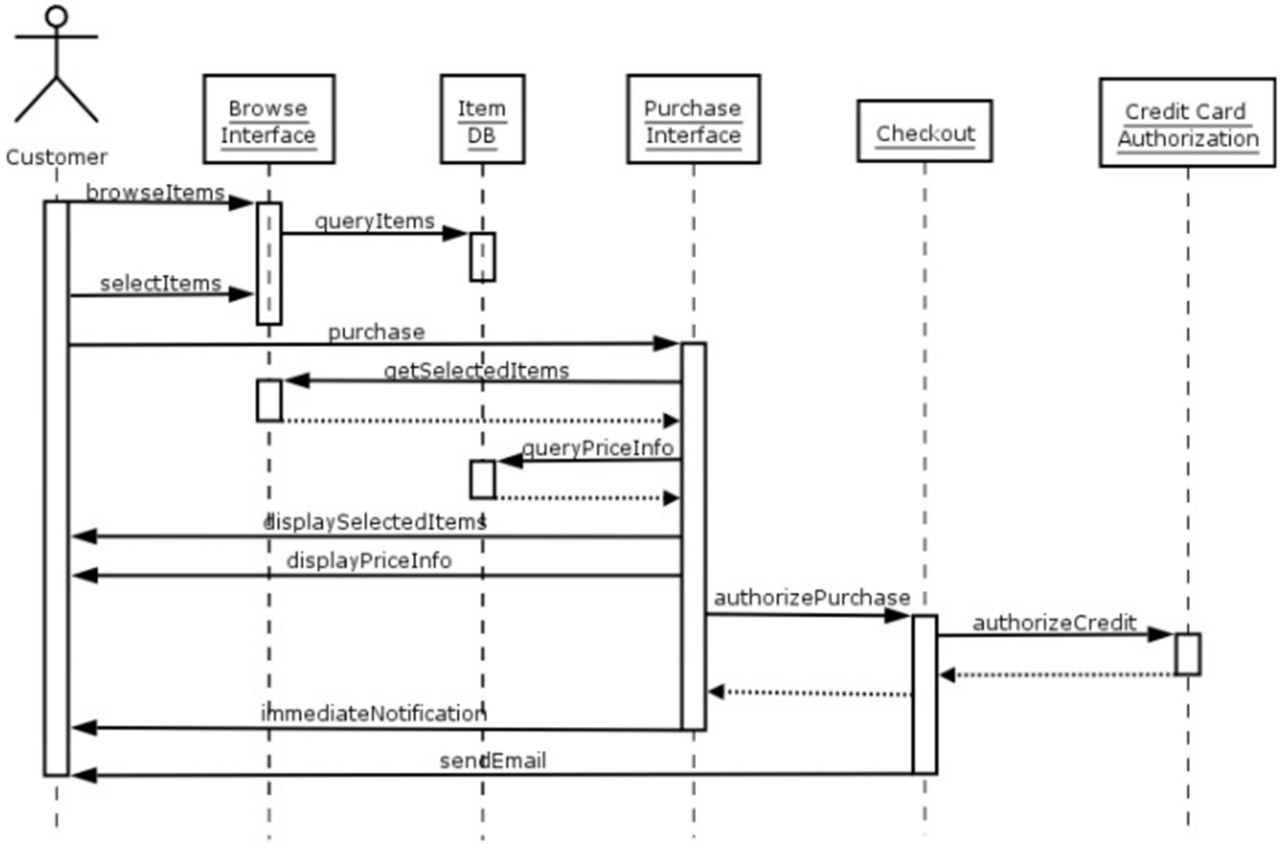
Sequence Diagram for Dispatching order and giving confirmation



H/w

1.Online Bookstore

Sequence diagram for online shopping



Sequence diagram for online shopping cloud service

Patterns for requirement Modeling

- Software patterns are a mechanism for **capturing domain knowledge**, that allows to **be reapplied when a new problem** is encountered.
- In some cases, the **domain knowledge is applied to a new problem within the same application domain**.
- The **domain knowledge captured by a pattern can be applied to a completely different application domain**.
- The o**riginal author of an analysis pattern does not "create" the pattern**, but, rather, *discovers* it as requirements engineering work is being conducted.
- Once the pattern has been discovered, it is documented by describing "explicitly the general problem to which the pattern is applicable, then prescribes solution,
- Also have consider the assumptions and constraints of using the pattern in practice, and often some other information about the pattern,
- Motivation and driving forces for using the pattern, discussion of the pattern's advantages and disadvantages, and references to some known examples of using that pattern in practical applications"
- The pattern can be reused when performing requirements modeling for an application within a domain.
- patterns are stored in a repository so that members of the software team can use search facilities to find and reuse them.
- Once an appropriate pattern is selected, it is integrated into the requirements model by reference to the pattern name

Discovering Analysis Pattern

- The requirements model is comprised of a wide variety of elements: scenario-based (use cases), data-oriented (the data model), class-based, flow-oriented, and behavioral.

- Each provides an opportunity to discover patterns that may occur throughout an application domain

- The most basic element in the description of a requirements model is the use case.

- A semantic analysis pattern (SAP) is a pattern that describes a small set of coherent use cases

- This use case implies a variety of functionality that would be refined and elaborated into a coherent set of use cases during requirements gathering and modeling.

- Regardless of how much elaboration is accomplished, the use cases suggest a simple, yet widely applicable SAP—the software-based monitoring and control of sensors and actuators in a physical system.

A Requirements Pattern Example: Actuator-Sensor

By considering One of the requirements of the SafeHome security function is the ability to monitor security sensors

Internet-based extensions to SafeHome will require the ability to control the movement of a security camera within a residence.

Actuator-Sensor that provides useful guidance for modeling this requirement within SafeHome software.

An abbreviated version of the Actuator-Sensor pattern, originally developed for automotive applications, follows.

Pattern Name. Actuator-Sensor

- **Intent**:Specify various kinds of sensors and actuators in an embedded system

- **Motivation**. Embedded systems usually have various kinds of sensors and actuators.

  These sensors and actuators are all either directly or indirectly connected to a control unit. Although many of the sensors and actuators look quite different,their behavior is similar enough to structure them into a pattern.

  The pattern shows how to specify the sensors and actuators for a system, including attributes and operations.

  The Actuator-Sensor pattern uses a pull mechanism for PassiveSensors and a push mechanism for the Active Sensors .

- **Constraints**

  • Each passive sensor must have some method to read sensor input and attributes that represent the sensor value.

  • Each active sensor must have capabilities to broadcast update messages when its value changes.

• Each active sensor should send a life tick , a status message issued within a specifi ed time frame, to detect malfunctions.

• Each actuator must have some method to invoke the appropriate response determined by the ComputingComponent .

• Each sensor and actuator should have a function implemented to check its own operation state.

• Each sensor and actuator should be able to test the validity of the values received or sent and set its operation state if the values are outside of the specifi cations.

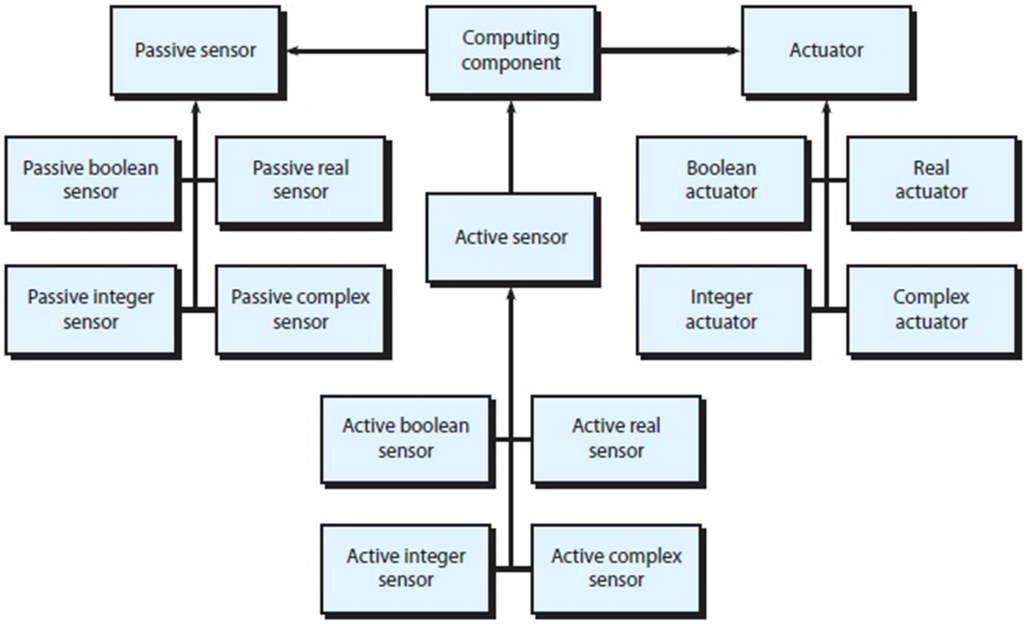
- **Applicability**. Useful in any system in which multiple sensors and actuators are present.

- **Structure**.
  Actuator, PassiveSensor, and ActiveSensor are abstract classes and denoted in italics.

  There are four different types of sensors and actuators in this pattern.
  The Boolean ,
  Integer , and
  Real classes represent the most common types

  The complex classes are sensors or actuators that use values that cannot be easily represented in terms of primitive data types, such as a radar device.

  Nonetheless, these devices should still inherit the interface from the abstract classes since they should have basic functionalities such as querying the operation states.


Passive sensor
Computing component
Actuator
Passive boolean sensor
Passive real sensor
Active sensor
Boolean actuator
Real actuator
Passive integer sensor
Passive complex sensor
Integer actuator
Complex actuator
Active boolean sensor
Active real sensor
Active integer sensor
Active complex sensor