Brunel University London

Software Development and Management
CS2002

Dr Giuseppe Destefanis
giuseppe.destefanis@brunel.ac.uk
@GiuseppeDes

Lecture 3

UML and Use Case diagrams

In this lecture

- Introduction to UML
- Use case diagrams

At the end of this lecture:

- You should be able to define UML and Use Case Diagrams
- You should be able to identify actors and Use cases from a problem statement.

What is a model?

- It is an abstraction of the system used to specify the structure and the behaviour
- A model capture all the important aspects of the objects we are modelling from a specific point of view, while omitting others (for example a floorpan)
- A model is expressed by a formalism which simplify its use and comprehension

Property of a model

- A model provides a representation which minimizes complexity
- A system is described by multiple views, which all together give us a complete vision of a model

Property of a model (II)

- A model contains the knowledge of the “what” and “how” of a system. It is used as a communication tool and for documenting the system under development
- It is fundamental for a collaborative process, such as software development

Unified Modeling Language

UML is a standardized modelling language consisting of an integrated set of diagrams, developed to help system and software developers for **specifying, visualizing, constructing, and documenting** the artifacts of software systems.

UML

- The UML represents a **collection** of **best engineering practices** that have proven **successful** in the modelling of large and complex systems.
- The UML is a very important part of developing object oriented software and the software development process.
- The UML uses mostly graphical notations to express the design of software projects.
- Using the UML helps project teams communicate, explore potential designs, and validate the architectural design of the software.

Why UML?

- As the strategic value of software increases for many companies, the industry looks for techniques to automate the production of software and to improve quality and reduce cost and time-to-market.

- These techniques include component technology, visual programming, patterns and frameworks.

Why UML?

- Businesses also seek techniques to manage the complexity of systems as they increase in scope and scale.
- In particular, they recognize the need to solve recurring architectural problems, such as physical distribution, concurrency, replication, security, load balancing and fault tolerance.

Why UML?

The primary goals in the design of the UML summarize by Page-Jones in Fundamental Object-Oriented Design in UML as follows:

1. Provide users with a ready-to-use, expressive visual modelling language so they can develop and exchange meaningful models.

2. Provide extensibility and specialization mechanisms to extend the core concepts.

3. Be independent of particular programming languages and development processes.

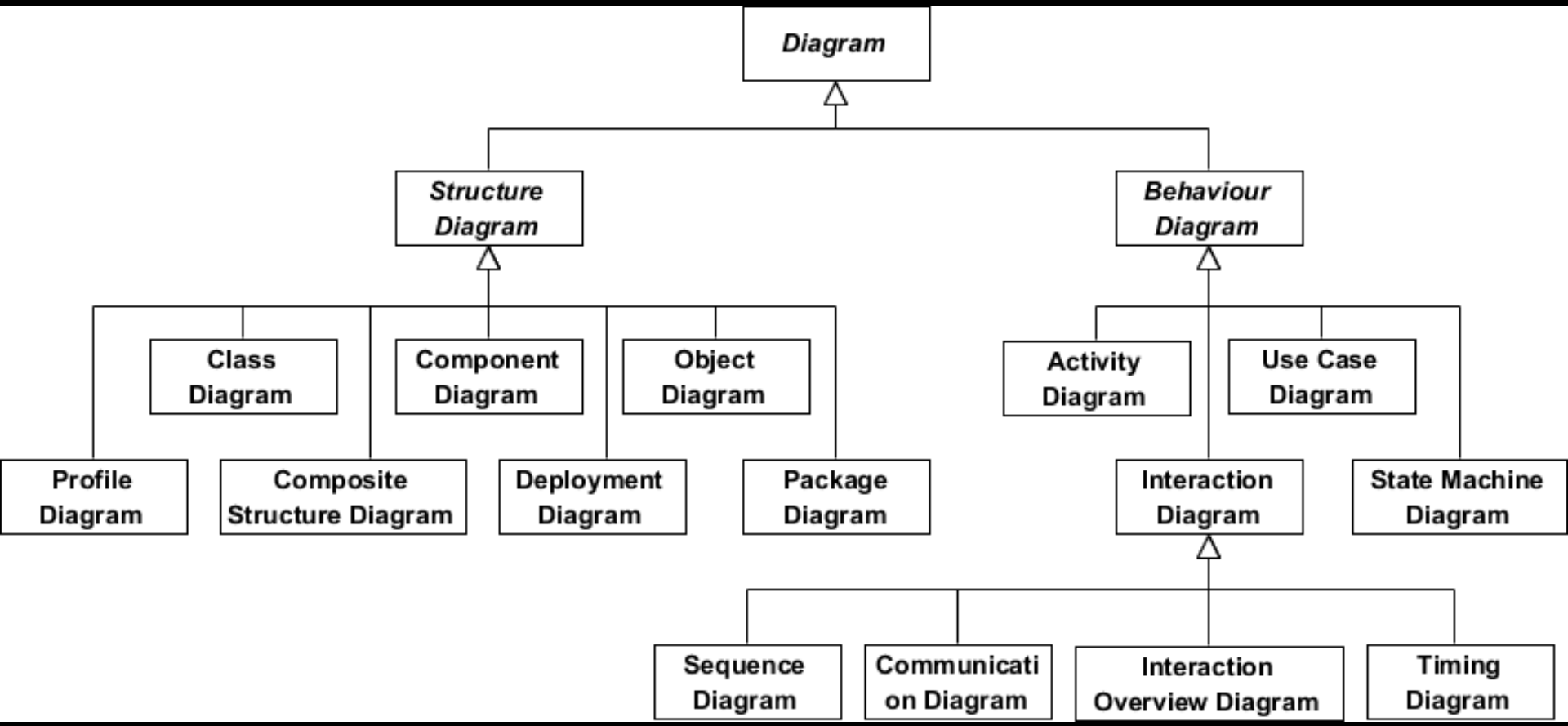
4. Provide a formal basis for understanding the modelling language.

5. Encourage the growth of the OO tools market.

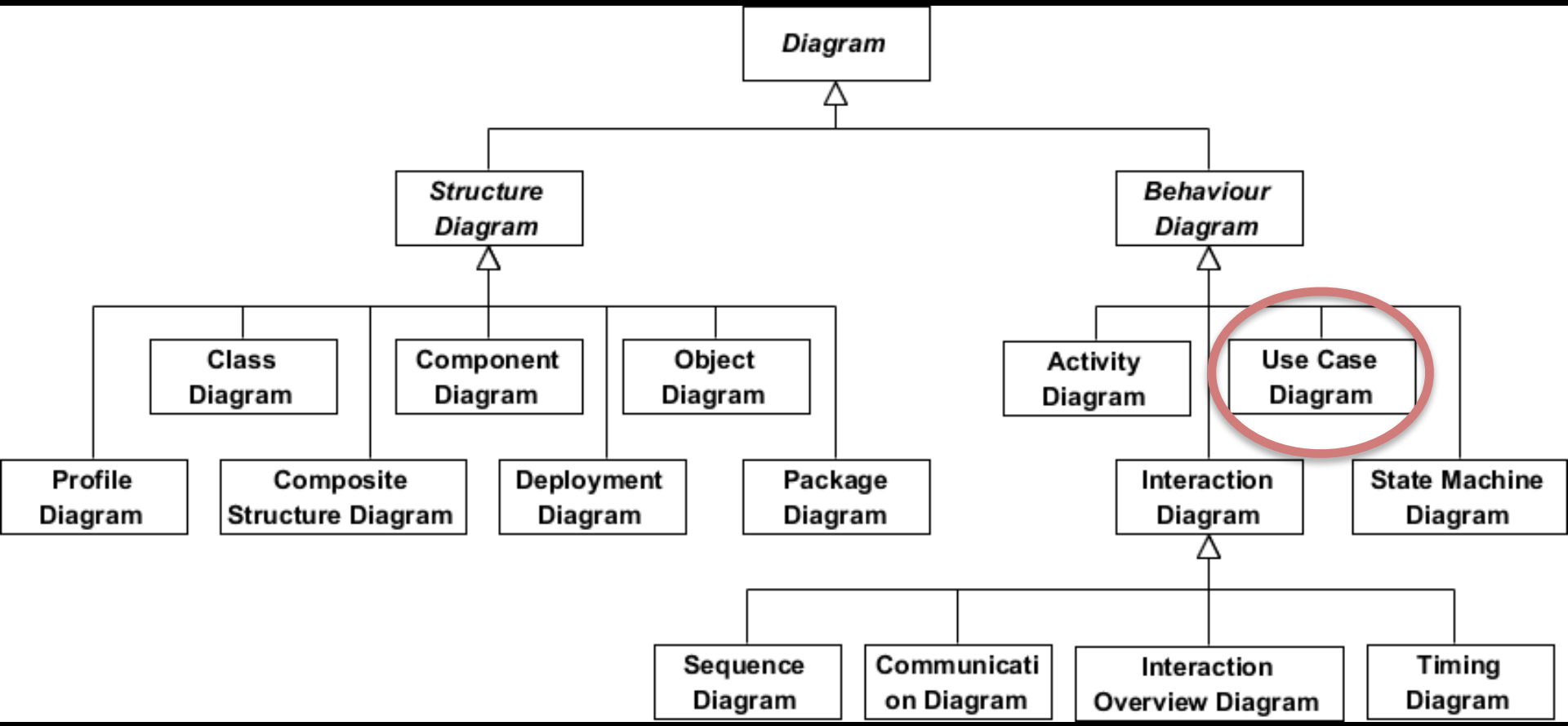
6. Support higher-level development concepts such as collaborations, frameworks, patterns and components.

7. Integrate best practices.

Different diagrams for different viewpoints



Different diagrams for different viewpoints



Use case Diagram

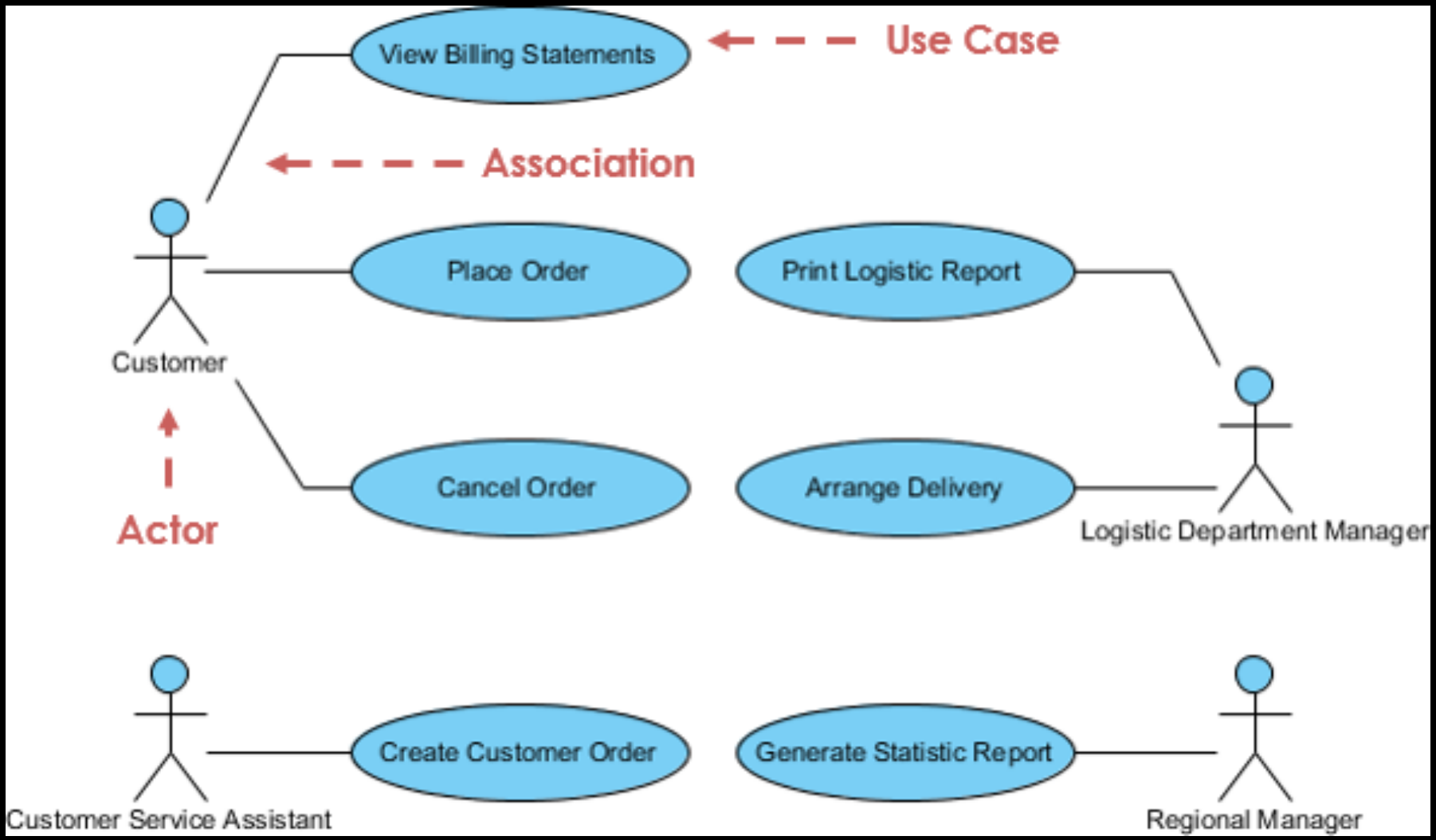
A use-case model describes a system's functional requirements in terms of use cases.

It is a model of the system's **intended functionality** (use cases) and its **environment** (actors).

Use cases enable you to relate what you need from a system to how the system delivers on those needs.

Use case Diagram (II)

- Think of a use-case model as a menu, much like the menu you'd find in a restaurant.
- By looking at the menu, you know what's available to you, the individual dishes as well as their prices. You also know what kind of cuisine the restaurant serves: Italian, Mexican, Chinese, and so on.
- By looking at the menu, you get an overall impression of the dining experience that awaits you in that restaurant. The menu “models" the restaurant's behaviour.


View Billing Statements
Use Case
Association
Place Order
Print Logistic Report
Customer
Cancel Order
Arrange Delivery
Logistic Department Manager
Actor
Create Customer Order
Generate Statistic Report
Customer Service Assistant
Regional Manager

Use case modelling

- A UML use case diagram is the primary form of system/software requirements for a new software program under development.
- Use cases specify the expected behaviour (what), and not the exact method of making it happen (how).
- Use cases once specified can be denoted both textual and visual representation (i.e. use case diagram).
- A key concept of use case modelling is that it helps us design a system from the end user's perspective. It is an effective technique for communicating system behaviour in the user's terms by specifying all externally visible system behaviour.

A use case diagram is simple

- It only summarizes some of the relationships between use cases, actors, and systems.
- It does not show the order in which steps are performed to achieve the goals of each use case.

Rule of thumb

- If your Use Case Diagram contain more than 20 use cases, you are probably misusing use case diagram.

Remember

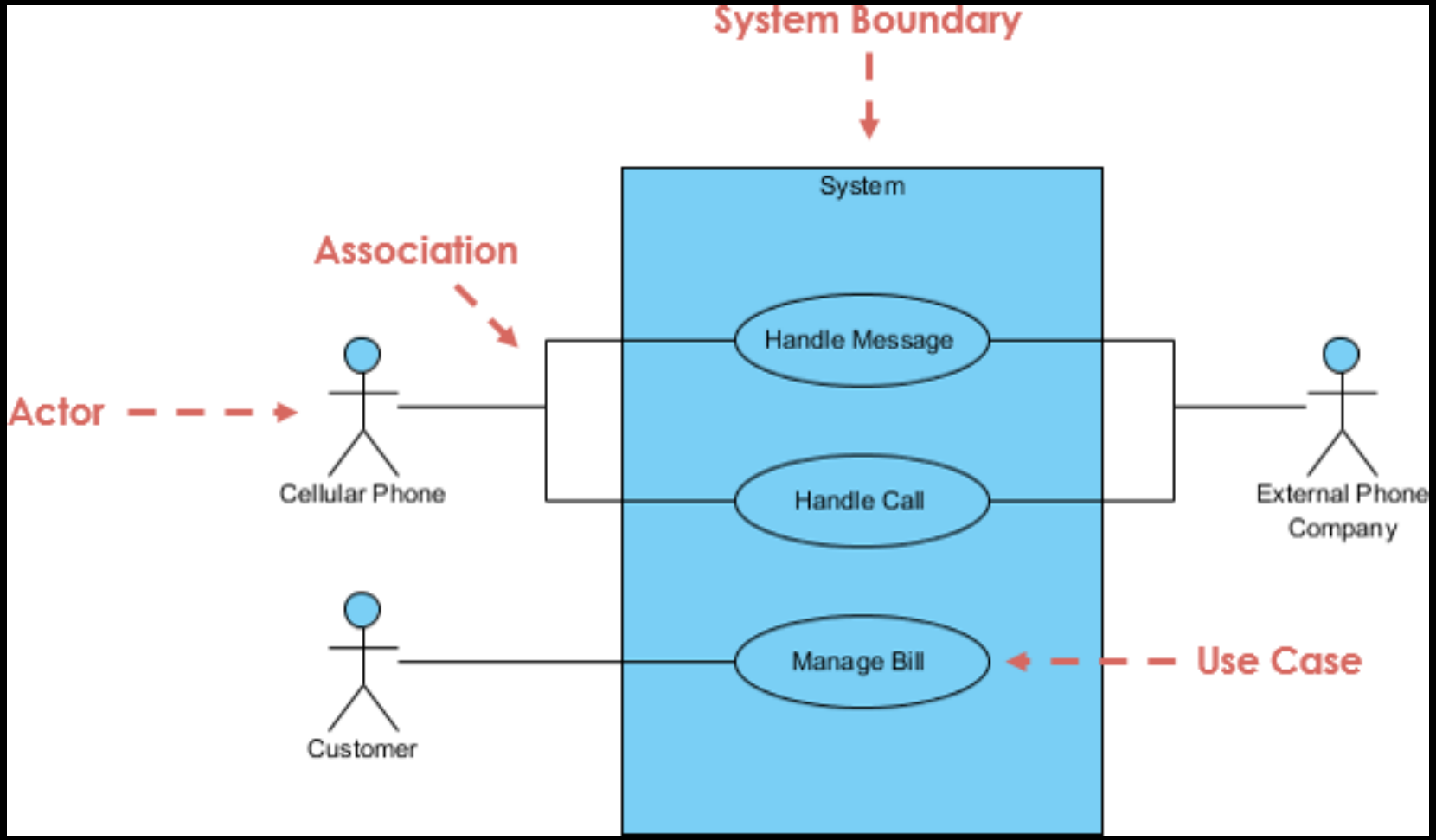
Use cases represent **only** the functional requirements of a system.

Other requirements such as business rules, quality of service requirements, and implementation constraints must be represented separately, again, with other UML diagrams.

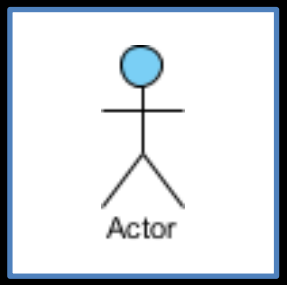
Purpose of Use Case Diagram

Use case diagrams are typically developed in the early stage of development and people often apply use case modelling for the following purposes:

- Specify the context of a system
- Capture the requirements of a system
- Validate a systems architecture
- Drive implementation and generate test cases
- Developed by analysts together with domain experts


System Boundary
System
Association
Handle Message
Actor
Cellular Phone
Handle Call
External Phone Company
Manage Bill
Use Case
Customer

Actor



- Someone which interacts with the use case (system function).
- Named by noun.
- Actor plays a role in the business
- Similar to the concept of user, but a user can play different roles
- For example:
    - A lecturer can be instructor and also researcher
    - plays 2 roles with two systems
- Actor triggers use case(s).
- Actor has a responsibility toward the system (inputs), and Actor has expectations from the system (outputs).

Use case

- System function (process - automated or manual)
- Named by verb + Noun (or Noun Phrase).
- i.e. Do something
- Each Actor must be linked to a use case, while some use cases may not be linked to actors.

Communication link

- The participation of an actor in a use case is shown by connecting an actor to a use case by a solid link.
- Actors may be connected to use cases by associations, indicating that the actor and the use case communicate with one another using messages.

Boundary of a system

- The system boundary is potentially the entire system as defined in the requirements document.
- For large and complex systems, each module may be the system boundary.
- For example, for an ERP system for an organization, each of the modules such as personnel, payroll, accounting, etc.
- can form a system boundary for use cases specific to each of these business functions.
- The entire system can span all of these modules depicting the overall system boundary

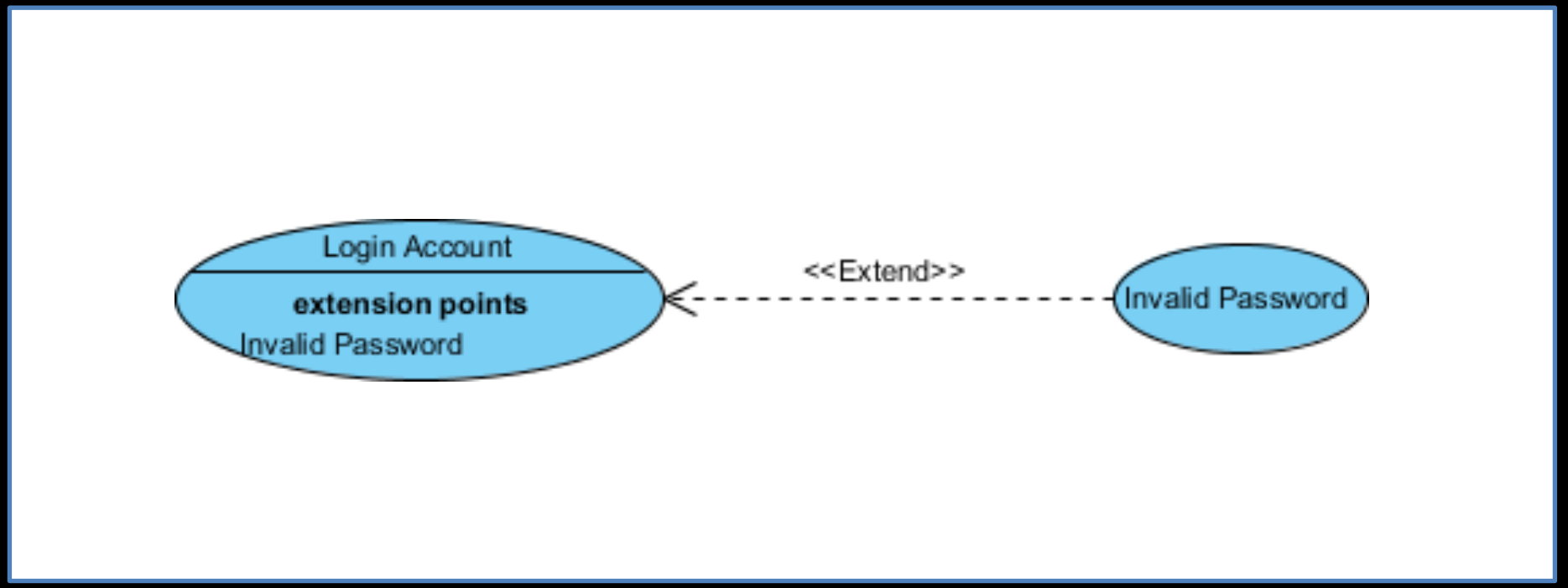
Use cases share different kinds of relationships

- Defining the relationship between two use cases is the decision of the software analysts.
- A relationship between two use cases is modelling the dependency between the two use cases.
- The reuse of an existing use case by using different types of relationships reduces the overall effort required in developing a system.

Extends

- Indicates that an "Invalid Password" use case may include the behaviour specified by base use case "Login Account".
- Depicted with a directed arrow having a dotted line. The tip of arrowhead points to the base use case and the child use case is connected at the base of the arrow.
- The stereotype **"<<extends>>"** identifies as an extend relationship

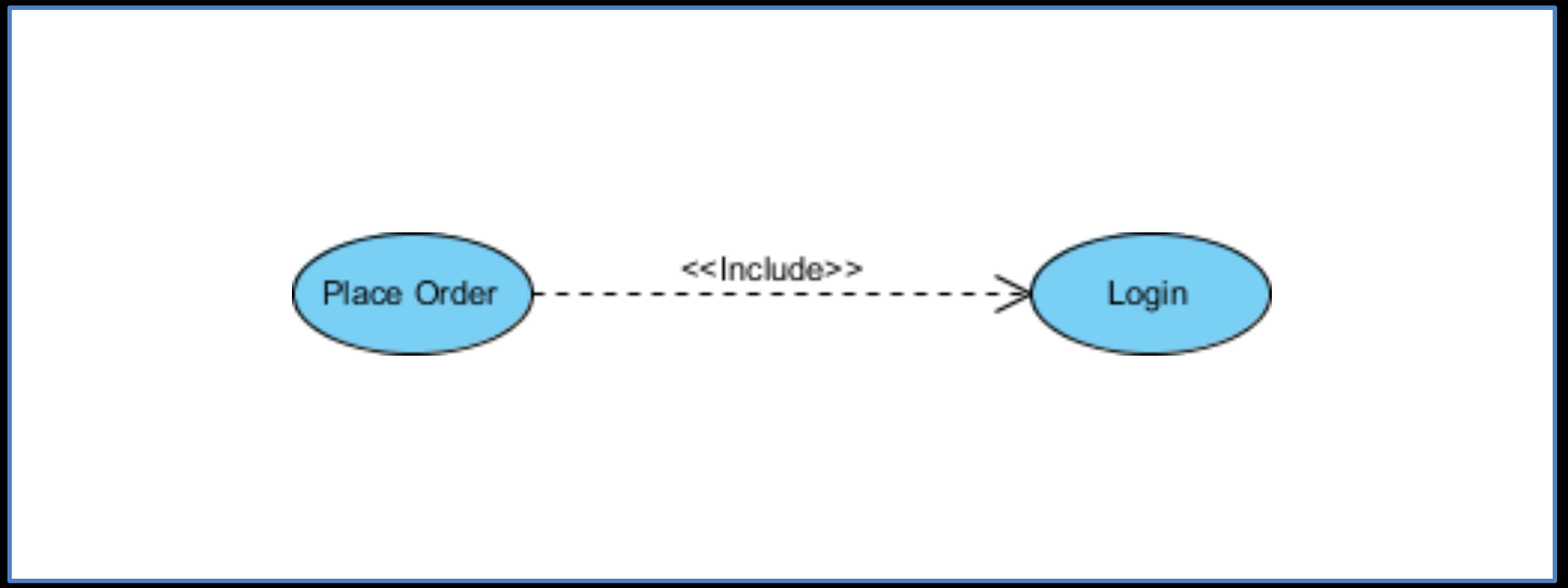
Extends


Login Account
extension points
Invalid Password
<<Extend>>
Invalid Password

Include

- When a use case is depicted as using the functionality of another use case, the relationship between the use cases is named as include or uses relationship.
- A use case includes the functionality described in another use case as a part of its business process flow.
- A uses relationship from base use case to child use case indicates that an instance of the base use case will include the behaviour as specified in the child use case.
- An include relationship is depicted with a directed arrow having a dotted line. The tip of arrowhead points to the child use case and the parent use case connected at the base of the arrow.
- The stereotype **"<<include>>"** identifies the relationship as an include relationship.

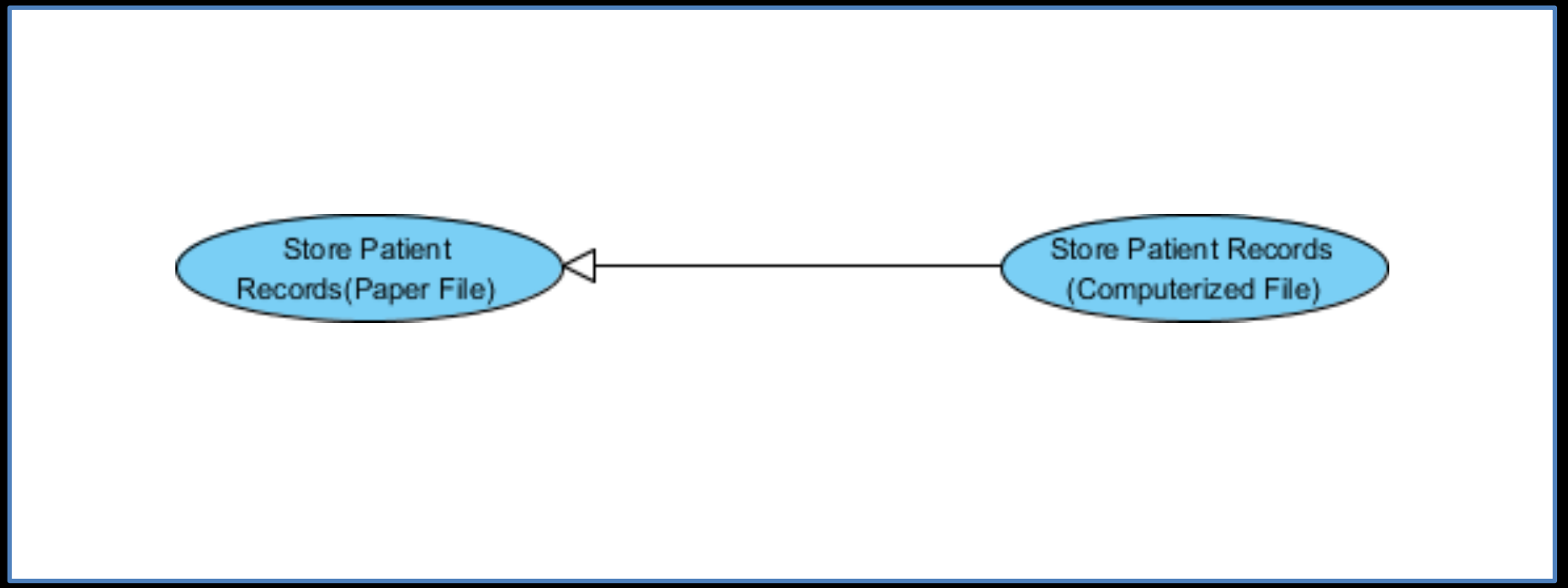
Include



Generalization

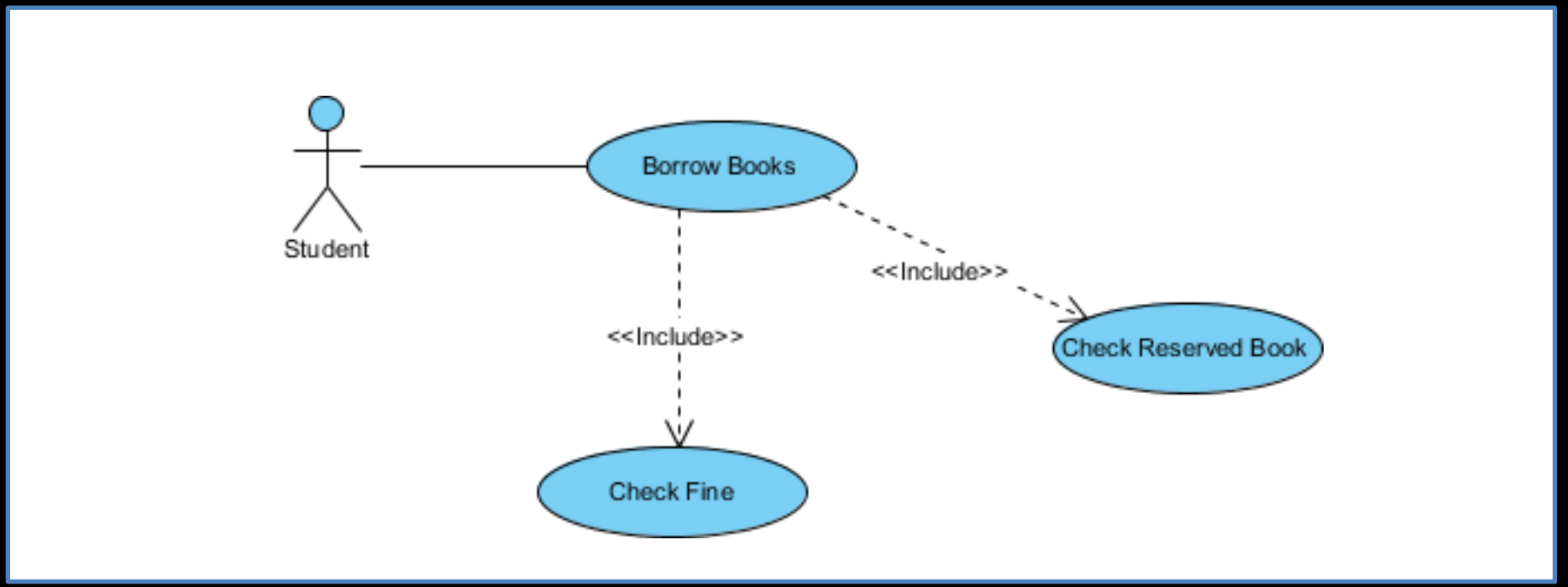
- A generalization relationship is a parent-child relationship between use cases.
- The child use case is an enhancement of the parent use case.
- Generalization is shown as a directed arrow with a triangle arrowhead.
- The child use case is connected at the base of the arrow. The tip of the arrow is connected to the parent use case.

Generalization

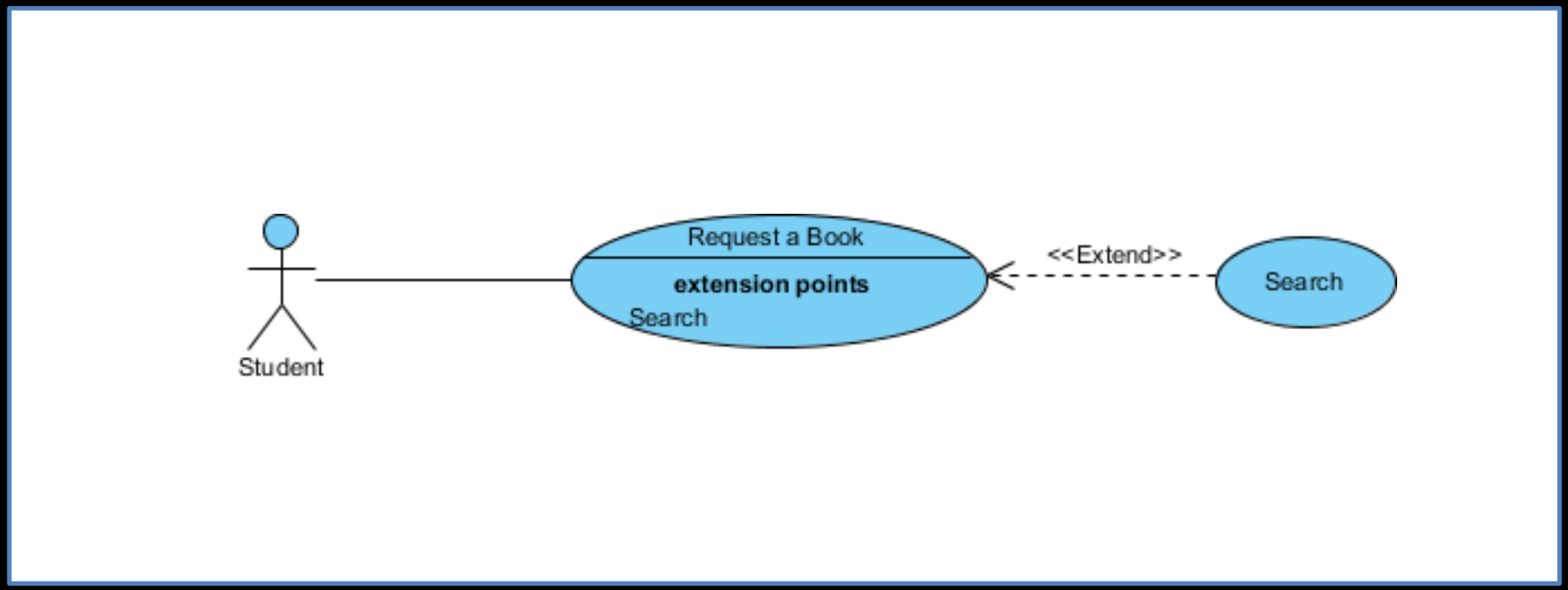


Use Case Example - Association Link

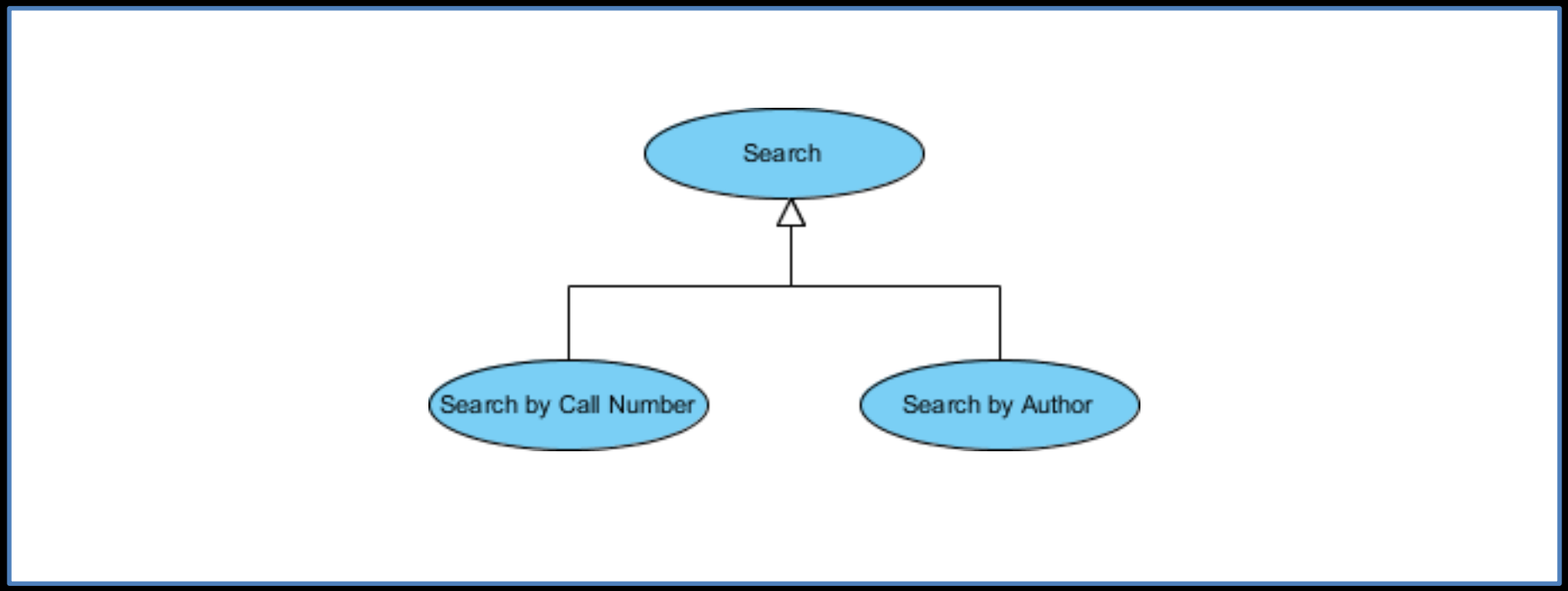
Use Case Example - Include Relationship



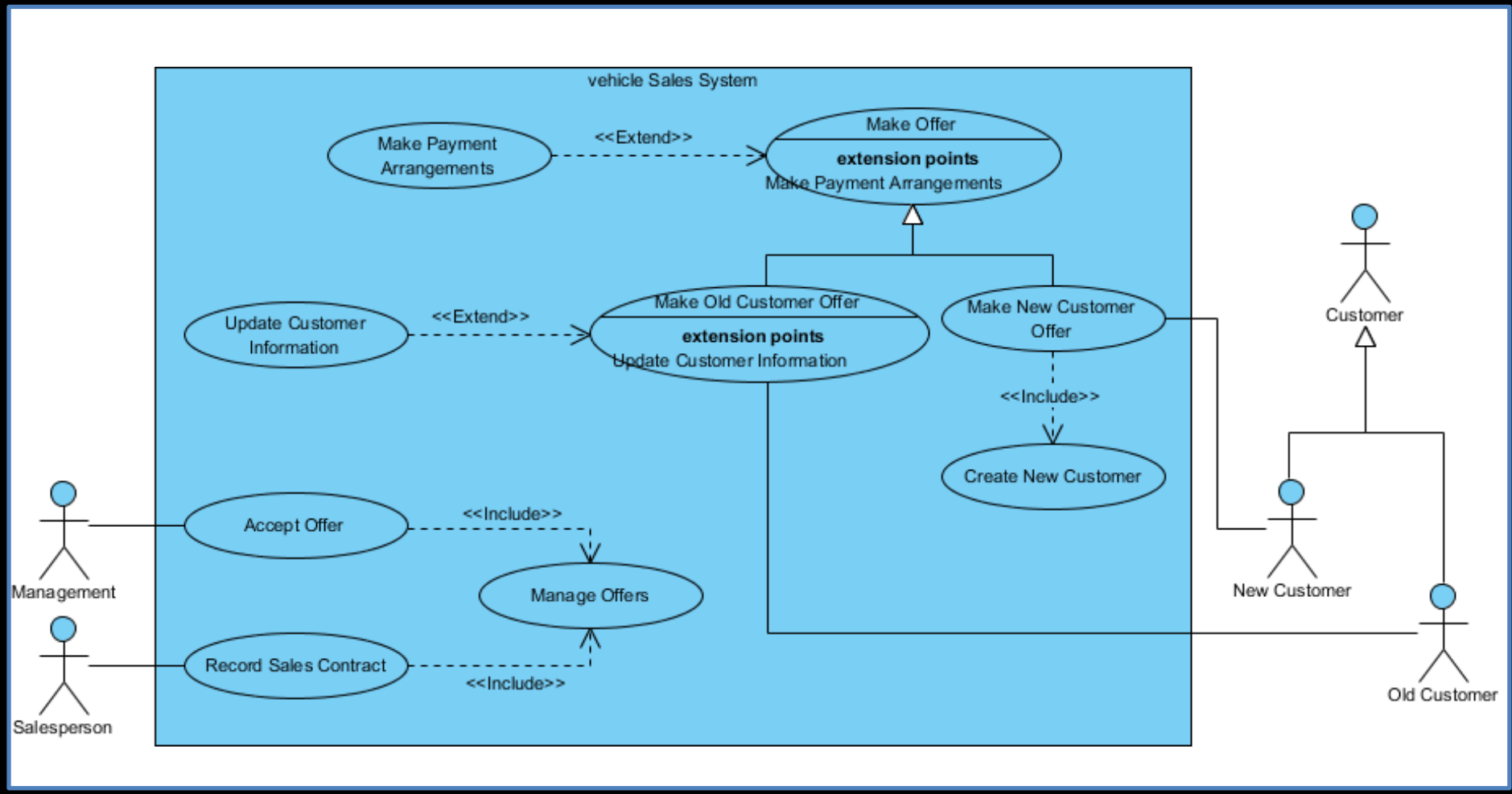
Use Case Example - Extend Relationship



Use Case Example - Generalization Relationship



Use Case Diagram - Vehicle Sales Systems



How to identify an Actor?

- Who uses the system?
- Who installs the system?
- Who starts up the system?
- Who maintains the system?
- Who shuts down the system?
- What other systems use this system?
- Who gets information from this system?
- Who provides information to the system?
- Does anything happen automatically at a present time?

How to identify Use Cases?

- What functions will the actor want from the system?
- Does the system store information? What actors will create, read, update or delete this information?
- Does the system need to notify an actor about changes in the internal state?
- Are there any external events the system must know about? What actor informs the system of those events?

Takeaway messages

- Always structure and organize the use case diagram from the perspective of actors.
- Use cases should start off simple and at the highest view possible. Only then can they be refined and detailed further.
- Use case diagrams are based upon functionality and thus should focus on the "what" and not the "how".

Reading material

- What is UML: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/
- Use case diagrams: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-use-case-diagram/