Everywhere you imagine. RENESAS

UNIFIED MODELING LANGUAGE (UML)

Renesas Design Viet Nam Co., Ltd.

Design Engineering Division

Presenter: Bao Nguyen – Software DTV Group

Modified date: Mar 03, 2008

Version 1.0

Confidential

Training Outline

1. Lecture

- On Mar 13 and half of Mar 14.
- Target: To give a brief introduction to UML.
- More details will be acquired in group's internal training.

2. Quiz

- Duration: About 45 minutes.
- To help review knowledge.

3. Q & A

- Please email me *bao.nguyen@rvc.renesas.com*
- Subject header: "[UML Training][Q&A]..."

Renesas Design Viet Nam Co., Ltd.

Lecture Outline (1)

Section 1 – INTRODUCTION TO UML

1. What is UML? Why to use UML?
2. Who needs UML?
3. Misconceptions about UML
4. Overview of UML Diagrams
5. Common modeling mistakes
6. UML Tools
7. Useful UML Modeling Tools
8. Useful UML Web Sites
9. Some Terminologies

Section 2 – MODEL CONCEPTS

1. What is model? Purposes of model?
2. Levels of models
3. What is in a model?

Lecture Outline (2)

Section 3 – UML VIEWS

1. UML Views – View Classification

- Structural classification
- Dynamic classification
- Model management

2. View Summary

- Static View
- Use Case View
- Interaction View
- State Machine View
- Activity View
- Physical View
  - Implementation View
  - Deployment View
- Model Management View

Renesas Design Viet Nam Co., Ltd.

Lecture Outline (3)

Section 4 – COMMON UML DIAGRAMS

1. Class Diagram
2. Use Case Diagram

Section 1 – INTRODUCTION TO UML

1. What is UML? Why to use UML?
2. Who needs UML?
3. Misconceptions about UML
4. Overview of UML Diagrams
5. Common modeling mistakes
6. UML Tools
7. Useful UML Modeling Tools
8. Useful UML Web Sites
9. Some Terminologies

What is UML? Why to use UML?

- UML = Unified Modeling Language
- UML is a standardized and general-purpose visual modeling language.
- UML captures decisions and understanding about systems to be constructed.
- UML consists of an integrated set of diagrams, that help system and software developers to accomplish the following tasks
  - Specification
  - Visualization
  - Architecture design
  - Construction
  - Simulation & Testing
  - Documentation

Who needs UML? (1)

- ***Modelers***

- Modelers try to describe the world as they see it - whether it is a system, a domain, an application or a world in their imagination.

---> UML is a tool for modelers to *document a particular aspect of some system*.

- ***Designers***

- Designers try to find out possible solutions, to compare, to trade off different aspects or to communicate approaches to constructive criticism.

---> UML is a tool for designers to *investigate a possible tactic or solution*.

Who needs UML? (2)

- ***Implementers***

- Implementers construct solutions using UML as the entire or a part of implementation approach.
- Many UML tools can now generate definitions for classes or databases, as well as application code, user interfaces or middleware calls.

---> UML is a tool *for implementers to understand their definitions*.

Misconceptions about UML (1)

- ***UML is not proprietary***

  - UML was originally conceived by Rational Software, but now it is owned by OMG (Object Management Group) and is open to all.

  - Many companies and individuals worked hard to produce UML 2. Good and useful information on UML is available from many sources.

- ***UML is not a process, or method or programming language***

  - UML encourages the use of modern object-oriented techniques and iterative life cycles. It is compatible with both predictive and agile control approaches.

  - However, despite the similarity of names, there is no requirement to use any particular “Unified Process”, and you may find such stuff inappropriate anyway.

Misconceptions about UML (2)

- ***UML is not difficult***
  - UML is big, but you don't need to use or understand it all.
  - You are able to select the appropriate diagrams for you needs and the level of details based on you target audience.

- ***UML is not time-consuming***
  - Properly used, UML cuts total development time and expenses as it decreases communication costs and increases understanding, productivity and quality.

Useful UML Websites

- www.omg.org (OMG is the owner of UML)
- www.uml.org
- www.uml-forum.com
- www.u2-partners.org

  (Latest proposals and news, etc... can be downloaded; Supported by UML 2.0; Partners: The leading group, called U2P - UML 2.0 Partners).
- http://community-ml.org/submissions.htm

  (Provide various proposals, including proposal called 3C (Clear, Clean, Concise); Supported by Community UML - One of the proposing groups, the communityUML).
- www.cs.york.ac.uk/puml/uml2_0.html

  pUML: A specific Web site for the precise UML group (pUML).
- www.2uworks.org

  A specific Web site for the 2U Consortium (Unambiguous UML).
- www.softdocwiz.com/UML.htm

  Kendall Scott's UML Dictionary provides look-up information of UML terms

UML Tools (1)

UML modeling tool, formerly known as a CASE (Computer-Aided Software Engineering) tool. A modeling tool assists the software's development by keeping track of all the software engineering symbols (such as those in UML) and it helps to do the following tasks

- ***Drawing UML diagrams***

Include class diagrams, use case diagrams and sequence diagrams

- ***Drawing UML notation correctly***

The tool draws a UML class as a box and a UML state as a rounded rectangle.
---> UML users do not have to to fool with getting the icon to look right.

- ***Organize the notation and the diagrams into packages***

With large projects, as the number of classes increase, UML users need help organizing their diagrams. Modeling tools help UML users organize by packages.

- ***Searching for specific elements in users' diagrams***

This is very helpful when UML users have a lot of diagrams with many classes, objects, associations, states and activities.

Renesas Design Viet Nam Co., Ltd.

UML Tools (2)

- ***Reverse engineering***

Some of the tools read UML users' object-oriented programming code and convert it into simple class diagrams. This saves time when UML users are modeling existing software.

- ***Model reporting***

UML users can support other developers with information about their models by asking the tool to generate a report.

- ***Generating code***

The big advantage of a UML modeling tool is the fast creation of some, but not all, of the code developers need for their software.

Some useful UML Modeling Tools (1)

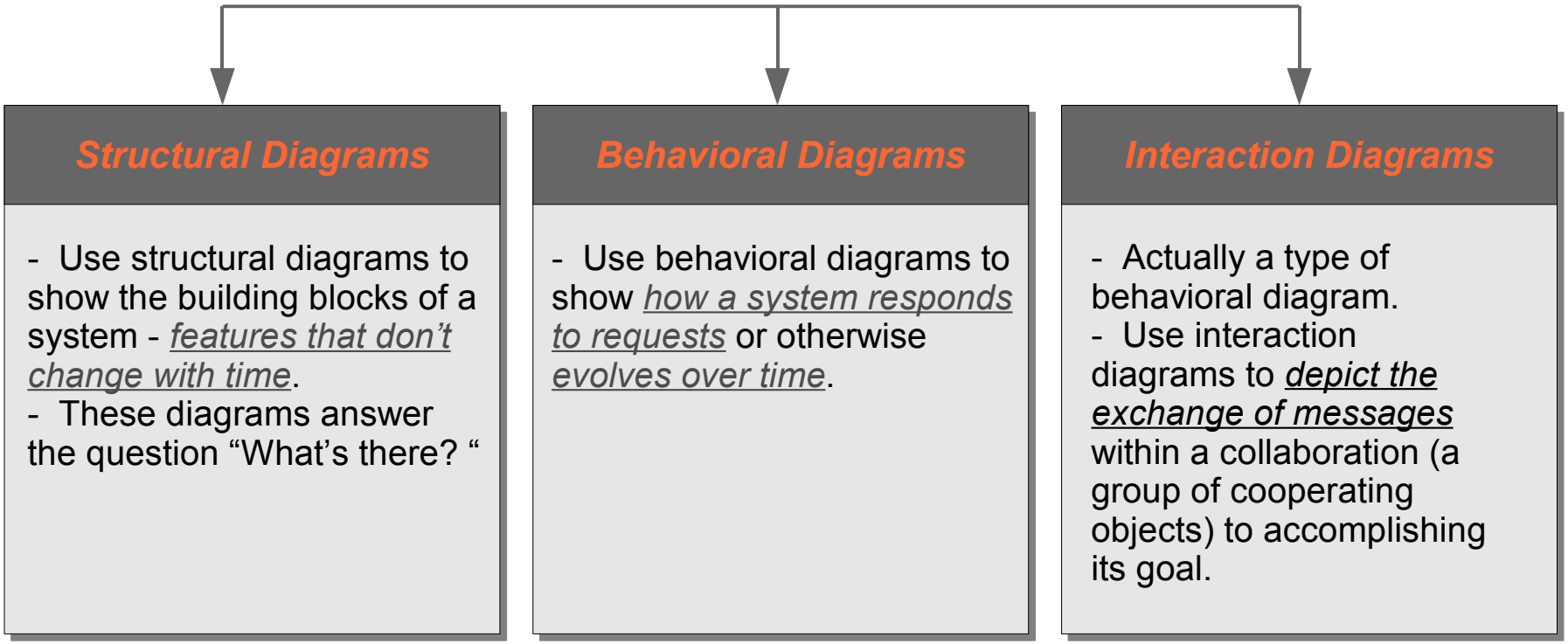
- **Argo/UML**
  - Produced by Tigris
  - Web site: argouml.tigris.org
  - A free tool, in Open Source community
  - A fast-growing and improving tool with supports of automated design wizards
- **Cittera**
  - Produced by: CanyonBlue
  - Web site: www.canyonblue.com
  - Cittera uses a Web-based repository for your models and will host your models. Choose Cittera if your development is distributed, mobile, and flexible.
- **Objecteering**
  - Produced by: Softeam
  - Web site: www.objecteering.com
  - The most powerful UML tool for constructing profiles.
  - A good choice for SPEM (Software Process Engineering Modeling), CWM (Common Warehouse Modeling) or EDOC (Enterprise Distributed Object Computing).
  - Especially handy if modifications are made to support special methodology.

Some useful UML Modeling Tools (2)

- **Rational Rose Suite**
  - Produced by: IBM
  - Web site: www.rational.com
  - Have a full suite of tools to support development, especially in areas such as requirements management and configuration management.
  - Rational has other UML tools, such as Rose R/T and Rational XDE, that are also worth looking at.
  - Choose Rational's tools if a full software development environment is needed.
- **Rhapsody**
  - Produced by: i-Logix
  - Web site: www.ilogix.com
  - A special tool for real-time or embedded-systems developer
- **Visio**
  - Produced by: Microsoft
  - Web site: www.microsoft.com/office/visio
  - Support UML drawing, code generation, reverse engineering and good notation coverage.

Renesas Design Viet Nam Co., Ltd.

Overview of UML Diagrams (1)



Renesas Design Viet Nam Co., Ltd.

Overview of UML Diagrams (2)

CATEGORY	TYPE OF DIAGRAM	PURPOSE
Structural Diagram	Class Diagram	Used to show real-world entities, elements of analysis and design, or implementation classes and their relationships.
	Object Diagram	Used to show a specific or illustrative example of objects and their links. Frequently used to indicate the conditions for an event, such as a test or an operation call.
	Composite Structure Diagram	Used to show the how something is made. Especially useful in complex structures-of-structures or component-based design.
	Deployment Diagram	Used to show the run-time architecture of the system, the hardware platforms, software artifacts (e.g. deliverable or running software items) and software environments, such as operating systems and virtual machines).
	Component Diagram	Used to show organization and relationships among the system deliverables.
	Package Diagram	Used to organize model elements and show dependencies among them

Overview of UML Diagrams (3)

CATEGORY	TYPE OF DIAGRAM	PURPOSE
Behavioral Diagram	Activity Diagram	Used to the show data flow and/or the control flow of a behavior and to captures workflow among cooperating objects.
	Use Case Diagram	Used to show the services that actors can request from a system.
	State Machine Diagram/ Protocol State Machine Diagram	Used to show the life cycle of a particular object, or the sequences an object goes through or that an interface must support.

Renesas Design Viet Nam Co., Ltd.

Overview of UML Diagrams (4)

CATEGORY	TYPE OF DIAGRAM	PURPOSE
Interaction Diagram	Overview Diagram	Used to show many different inter- action scenarios (sequences of behavior) for the same collaboration (a set of elements working together to accomplish a goal).
	Sequence Diagram	Used to focus on message exchange between a group of objects and the order of the messages.
	Communication Diagram	Used to focus on the messages between a group of objects and the underlying relationship of the objects.
	Timing Diagram	Used to show changes and their relationships to clock times in real-time or embedded systems work.

Overview of UML Diagrams (5)

- Since UML is very flexible, there are *various ways of categorizing the diagrams*.
- The following three categories are popular
  - ***Static diagrams***
    - Show the static features of a system.
    - This category is *similar to that of structural diagrams*.
  - ***Dynamic diagrams***
    - Show how a system evolves over time.
    - Contain UML state-machine diagrams and timing diagrams.
  - ***Functional diagrams***
    - Show the details of behaviors and algorithms
      --> How a system accomplishes the behaviors requested of it?
    - This category includes use-case, interaction and activity diagrams.

Common Modeling Mistakes (1)

- ***Using too few or too many diagram types***

Example

- Some developers use ONE diagram for every situation. They build class diagrams to capture classes and their static relationships, but also represent object interactions, data flows and system decompositions with those same class diagrams. Unfortunately, the class diagram does not capture that other stuff very well, but use case, sequence, state and activity diagrams are the best choice.
- Other developers use every single UML diagram whether they need to or not.

Some people are so proud of their UML knowledge ---> Show off their abilities !!!

---> Waste valuable time trying to decipher these extra diagrams without making any progress toward completing the project !!!

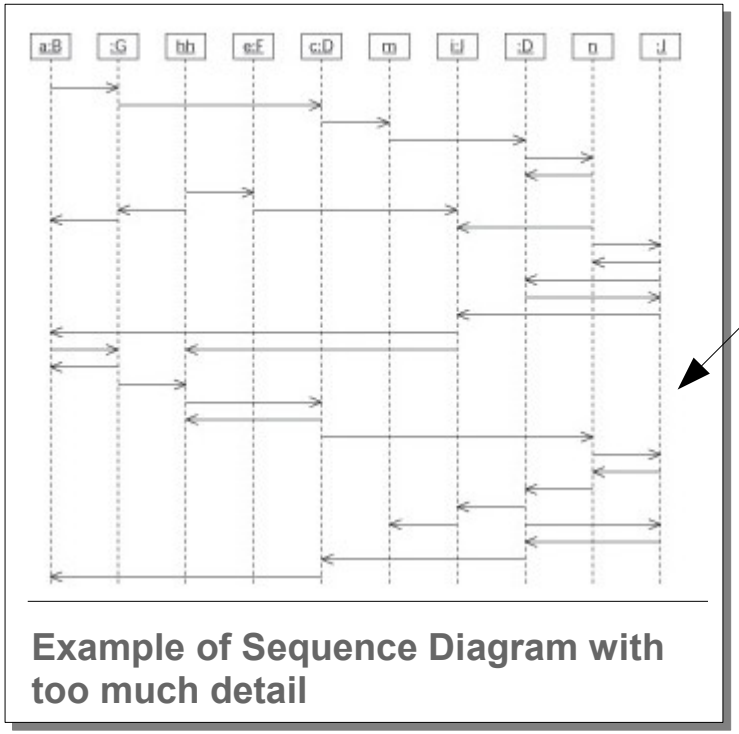
Tips

*** If maintaining an existing system,
-----> Class diagrams + Sequence diagrams + Deployment diagrams
*** If building real-time embedded systems,
-----> State diagrams + Sequence diagrams + Class diagrams

Renesas Design Viet Nam Co., Ltd.

Common Modeling Mistakes (2)

- ***Showing too much detail***



Example of Sequence Diagram with too much detail

Unable to update details or maintain this kind of diagram when requirements change !!!

TIP

Detail is NOT shown in the diagram

Risk EXISTS

Add more details

Renesas Design Viet Nam Co., Ltd.

Common Modeling Mistakes (3)

- ***Define the same thing twice***

- Two users use different words to mean the same thing.

Tips

* Look through diagrams to find classes with similar attributes, operations and associations.
* If a couple of similar classes is found, users and other developers should be questioned, also ask for examples of these similar classes from users.
* If they turn out to be the same, a single name for the class should be chosen and used.

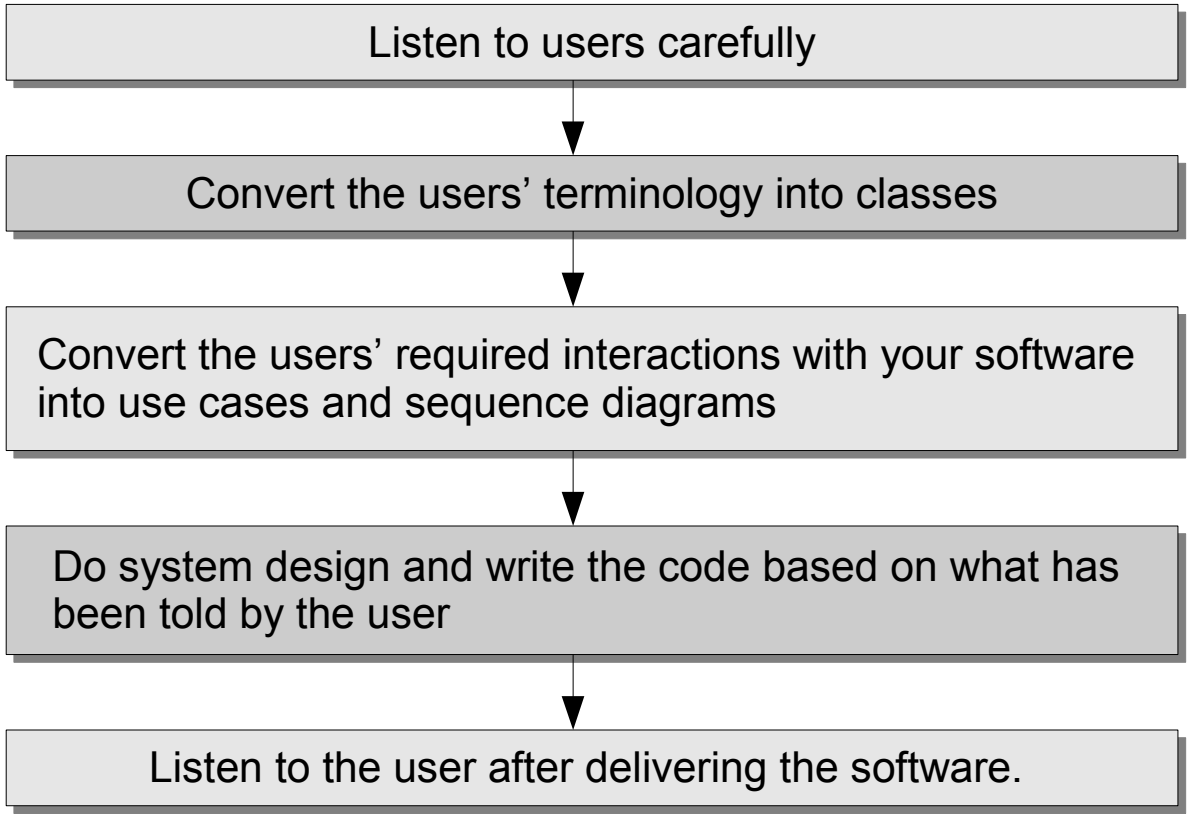
Common Modeling Mistakes (4)

- ***Not listen to the users***

- Rather than listen to what a user needs from a software application, some developers are too busy thinking about how they are going to write their next program.
- In the end, the software does not meet the needs of the user.
- When users ask for a better system, the new developers get completely confused because they can't relate what the user is saying to anything in the program code.

Common Modeling Mistakes (5)

- ***Tip for good listening to users***



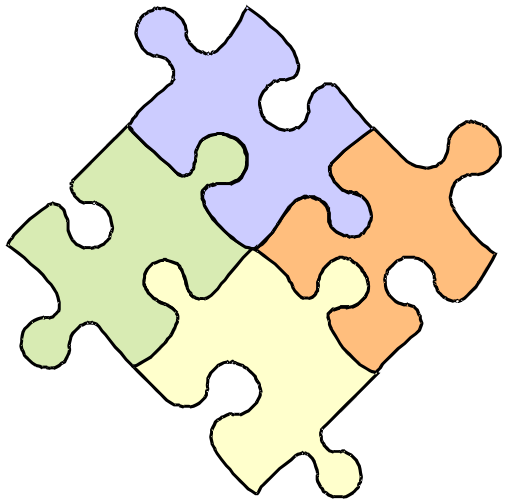
Renesas Design Viet Nam Co., Ltd.

Some terminologies (1)

- ***Class***
- A family of objects with similar structure and behavior.
- ***Object***
- Refer to something useful that has identity, structure and behavior.
- ***Abstraction***
- Describe the essence of an object for a purpose.
- ***Encapsulation***
- Which is needed to know to use an object.
- ***Information Hiding***
- Keep it simple by hiding the details.
- ***Aggregation***
- The whole object or some parts of the whole object should be told.

Some terminologies (2)

- ***Generalization***
  - What is common among these objects?
- ***Specialization***
  - What is the difference among this particular object and the others?
- ***Inheritance***
  - Specialized objects inherit the common features of generic objects.



Renesas Design Viet Nam Co., Ltd.

Section 2 – MODEL CONCEPT

1. What is model? Purposes of models?
2. Levels of models
3. What is in a model?

What is model? Purposes of models

- Model is a simplification of reality.
- ***Purposes of models***
  - To capture and precisely state requirements and domain knowledge so that all stakeholders may understand and agree on them.
  - To think about the design of a system.
  - To capture design decisions in a flexible form separate from the requirements.
  - To generate usable work products.
  - To organize, find, filter, retrieve, examine and edit information about large systems.
  - To explore multiple solution economically.
  - To master complex systems.

Renesas Design Viet Nam Co., Ltd.

Levels of models

Models take on different forms for various purposes and appear at *different levels* of abstraction. Thus, the amount of detail in the model must be adapted to one of the following purposes

- Guides to the thought process.
- Abstract specifications of the essential structure of a system.
- Full specifications of a final system.
- Exemplars of typical or possible systems
- Complete or partial descriptions of systems

What is in a model? (1)

- ***Semantics and presentation***

Models have two major aspects: *semantic information* (semantics) and *visual presentation* (notation)

Semantic information

- Semantic aspect captures the meaning of an application as a network of logical constructs (e.g. classes, associations, states, use cases and messages)
- Semantic model elements convey the meaning of the model.
- Semantic modeling elements are used for code generation, validity checking, complexity metrics, etc...
- The semantic information is often called *the model.*
- The visual presentation shows semantic information in a form that can be seen, browsed and edited by humans.

Visual presentation

Presentation elements carry the visual presentation of the model. They do not add meaning, but they do organize the presentation to emphasize the arrangement of the model in a usable way.

What is in a model? (2)

- ***Context***

- Models are themselves artifacts in a computer system, and they are used within a larger context that gives them their full meaning.

- Context includes the internal organization of the model, annotations about the use of each model in the overall development process, a set of defaults and assumptions for element creation and manipulation and a relationship to the environment in which they are used.

Section 3 – UML VIEWS

1. UML Views – View Classification

- Structural classification
- Dynamic classification
- Model management

2. View Summary

- Static View
- Use Case View
- Interaction View
- State Machine View
- Activity View
- Physical View
  - Implementation View
  - Deployment View
- Model Management View

UML Views – View Classification

- A view is simply a subset of UML modeling constructs that represents one aspect of a system.
- ***View classification***
  - ***Structural classification***
    - Describe the things in the system and their relationships to other things.
    - Include Static View, Use Case View, Implementation View and Deployment View.
  - ***Dynamic classification***
    - Describe the behavior of a system over time.
    - Include State Machine View, Activity View and Interaction Views. (Sequence Diagrams and Collaboration Diagrams)
  - ***Model management classification***
    - Describe the organization of the models themselves into hierarchical units.
    - The Model Management View crosses the other views and organizes them for development work and configuration control.

Renesas Design Viet Nam Co., Ltd.

Static View

- Diagram: *Class Diagram*
- Main concepts: *class, association, generalization, dependency, realization, interface*
- Models concepts in the application domain and internal concepts invented as part of the implementation of an application.
- This view does *not describe the time-dependent behavior of the systems*.
- Main constituents are below
  - Classes
  - Relationship: Association, Generalization
  - Dependency: Realization, Usage
- Classes can be described at various levels of precision and concreteness. In the early stages of design, the model captures more logical aspects of the problem. In the later stages, the model also captures design decisions and implementation details.

Use Case View

- Diagram: *Use Case Diagram*
- Main concepts: *use case, actor, association, extend, include, use case generalization*
- Models the functionality of the system as perceived by outside users, called *actors.*
- Purpose of this view is to list actors and use cases and show which actors participate in each use case.
- Use cases can be described at various levels of detail. They can be factored and described in terms of other, simpler use cases.
- A use case is *implemented as collaboration in the interaction view.*

Interaction View (1)

- Describes sequences of message exchanges among roles that implement behavior of a system.
- A classifier role is the description of an object that plays a particular part within an interaction, as distinguished from other objects of the same class.
- This view *shows the flow of control across many objects*.
- Both sequence diagram and collaboration diagram show interactions but they emphasize different aspects.
- A sequence diagrams shows *time sequence* as a geometric dimension, but the *relationships among roles are implicit*.
- A collaboration diagrams shows *the relationships among roles* geometrically and relates messages to the relationships, but the *time sequences are less clear*.
- Each diagram should be used when its main aspect is the focus of attention.

Interaction View (2)

Sequence Diagram

- Main concepts: *interaction, object, message, activation*
- Shows a set of *messages* arranged in time sequence.
- A sequence diagram can show
  - a *scenario* which is an individual history of a transaction
  - the *behavior sequence of a use case*

Collaboration Diagram

- Main concepts: *collaboration, interaction, collaboration role, message*
- Models the *objects* and links that are meaningful within an *interaction*
- A collaboration diagram shows the implementation of an operation.

State Machine View

- Diagram: *State chart Diagram*
- Main concepts: *state, event, transition, action*
- Models the possible life histories of an *object* of a class.
- Contains *states* connected by *transitions*.
- Each state models a period of time during the life of an object during which it satisfies certain conditions.
- When an *event* occurs, it may cause the firing of a *transaction* that takes the object to a new state.
- When a transition *fires*, an *action* attached to the transition may be executed.
- This view may be used to describe user interfaces, device controllers and other reactive subsystems.

Activity View

- Diagram: *Activity Diagram*
- Main concepts: *state, activity, completion transition, fork, join*
- Is a variant of a state machine that shows the computational activities involved in performing a calculation.
- An *activity state* represents an *activity* which is a work flow step or the execution of an *operation.*
- An activity graph describes both sequential and concurrent groups of activities.
- An activity diagram models the real-world work flows of a human organization.

---> Such business modeling is a major purpose of activity diagrams.

- Helpful in understanding the high-level execution behavior of a system, without getting involved in the internal details of message passing required by a collaboration diagram.

Physical View (1)

- Models the implementation structure of an application itself, such as its organization into components and its deployment onto run-time nodes.
- These views provide an opportunity to map classes onto implementation components and nodes.

Physical View (2)

Implementation View

- Diagram: *Component Diagram*
- Main concepts: *component, interface, dependency, realization*
- Models the components in a system – that is, the software units from which the application is constructed as well as the dependencies among components so that the impact of a proposed change can be assessed.
- It also models the assignment of classes and other model elements to components.

Physical View (3)

Deployment View

- Diagram: *Deployment Diagram*
- Main concepts: *node, component, dependency, location*
- Represents the arrangement of run-time *component* instances on *node* instances.
- A node is a run-time resource, such as a computer, device or memory.
- This view allows the consequences of distribution and resource allocation to be assessed.
- Two different levels of deployment diagrams
  - ***Descriptor-level deployment diagram*** --> Shows the kinds of nodes in the system and the kinds of components they hold.
  - ***Instance-level deployment diagram*** --> Shows the individual nodes and their links in a particular version of the system.

Renesas Design Viet Nam Co., Ltd.

Model Management View

- Diagram: *Class Diagram*
- Main concepts: *package, subsystem, model*
- Models the organization of the model itself
- A *model* comprises a set of *packages* that hold *model elements*, such as *classes, state machines and use cases.*
- Packages may contain other packages and are units for manipulating the contents of a model, as well as units for access control and configuration control.
- Each model element is owned by one package or one other element.
- A model is a complete description of a system at a given precision from one viewpoint.
- A *subsystem* is another special package. It represents a portion of a system, with a crisp interface that can be implanted as a distinct component.

Renesas Design Viet Nam Co., Ltd.

Section 4 – COMMON UML DIAGRAMS

1. Class Diagram

2. Use Case Diagram

1. CLASS DIAGRAM

Overview

- Class diagrams are the most common diagram found in modeling object-oriented systems.
- A class diagram shows a set of classes, interfaces and collaborations and their relationships.
- Class diagrams are used to illustrate the static design view of a system, involving modeling the vocabulary of the system, modeling collaborations or modeling schemata.
- To document these information, the Class diagram includes attributes, operations, stereotypes, properties, associations and inheritance.

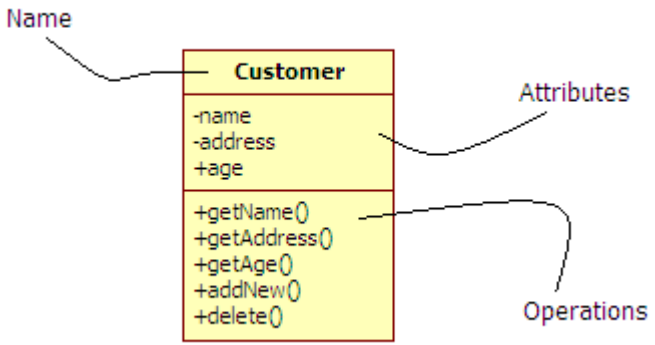
Class Diagram Components

- ***Classes and objects***
  - Attributes
  - Operations
  - Visibility
- ***Relationships***
  - Association
  - Generalization
  - Flow
  - Dependency
    - *Realization*
    - *Usage*
- ***Interfaces***
  - Provided interface
  - Required interface

Renesas Design Viet Nam Co., Ltd.

Class

- A class is a description of a set of objects that share the same attributes, operations, relationships and semantics.
- Classes are used
  - To capture the vocabulary of the system which is being developed.
  - To represent software things, hardware things, and even things that are purely conceptual.



Class notation with 3 compartments: name, attributes, and operations

Relationship

- Relationship is used to model the ways that things can connect to one another, either logically or physically.
- Relationships among classifiers are *association, generalization, flow*, and various kind of dependency, including *realization and usage*.

Associations

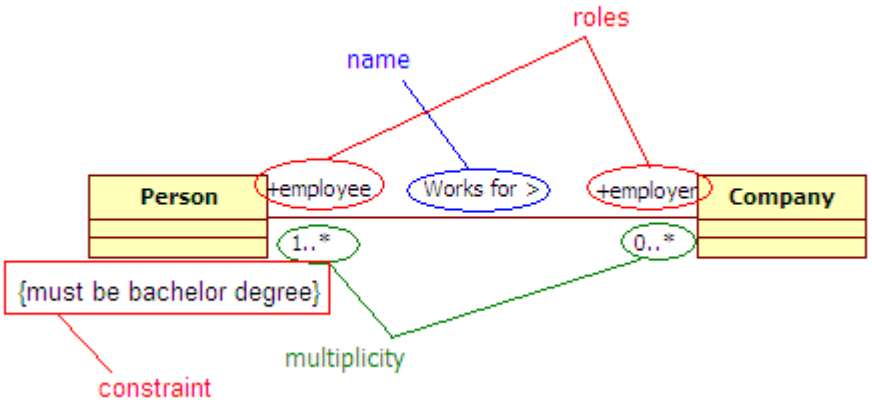
- An association is a structural relationship that specifies that objects of one thing are connected to objects of another.
- Associations can connect more than two classes, then are called n-ary associations.
- An association is depicted by such details as name, multiplicity, role, constraint, qualifier, directional navigation... (it is not necessary to put all these details on each and every association in our diagrams).

Class Association Details

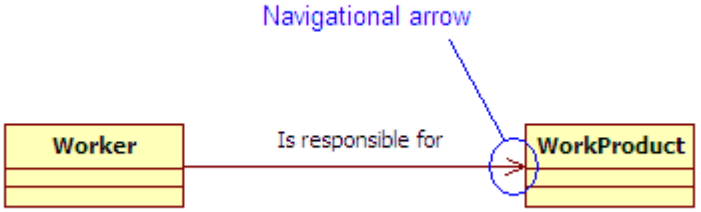
- ***Name***
  - Used to describe the nature of the relationship and how objects of one type (class) relate to objects of another type (class).
  - Accompanied by *name-direction arrow*.
- ***Role***
  - Describe how an object participates in the association.
  - The same class can play the same or different roles in other associations.
- ***Multiplicity***
  - Define the number of participating objects.
- ***Constraint***
  - Specify the rule(s) which underlying links of the association must follow.
- ***Directional navigation***
  - Indicate the direction of allowable communication with an arrowhead at one end of the association.

Renesas Design Viet Nam Co., Ltd.

Class Association Details - EXAMPLE



Association name, role, constraint, and multiplicity

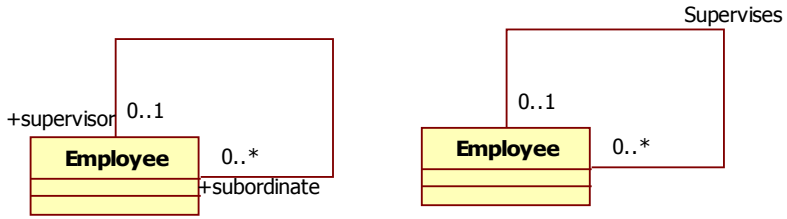


Association navigation

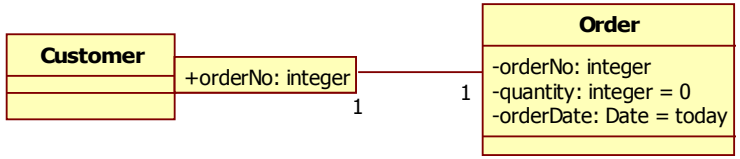
Renesas Design Viet Nam Co., Ltd.

Extended Associations

- ***Reflexive association***
  - Indicate objects in the same class can be related to one another.
- ***Qualified association***
  - Simplify the navigation across complex associations by providing keys called *qualifiers* to narrow the selection of objects



Reflexive association *presented by using role names (left) and association name (right)*



Qualified association

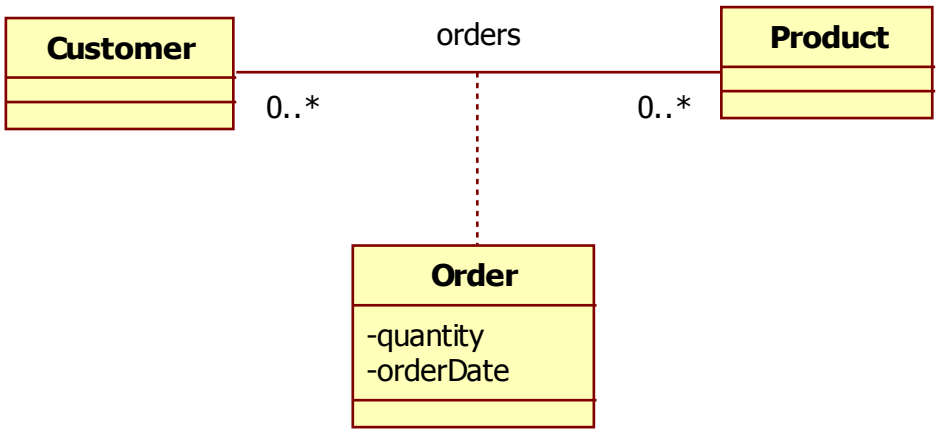
The Customer uses orderNo as a qualifier to look up an Order

(!) Typically, the qualifier is an attribute of the class on the opposite end of the association, so make certain that the 2 names and data types agree

Renesas Design Viet Nam Co., Ltd.

Extended Associations – ASSOCIATION CLASS

- The association between 2 classes might have properties – it's time to use the concept of association class.
- An association class encapsulates information about an association.

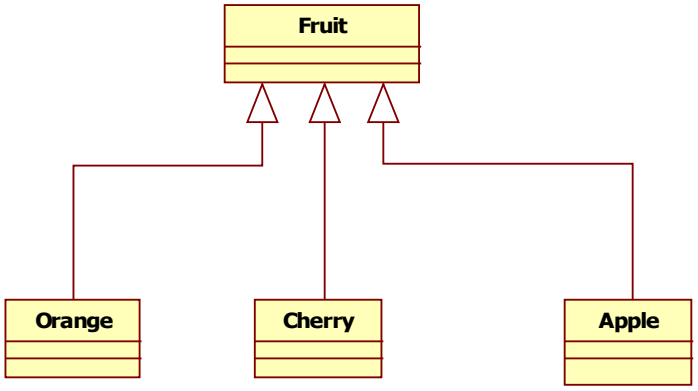


Association class *notation is rendered as a class symbol attached by a dashed line to an association line*

Renesas Design Viet Nam Co., Ltd.

Generalization

- Generalization is a relationship between a general kind of thing (called the *superclass* or *parent*) and a more specific kind of thing (called the *subclass* or *child*).
- The child class or subclass can inherit attributes and operations from the parent class or superclass.
- Generalizations are used to show parent – child relationships.
- Generalization is not association – so that there is no need for multiplicity, role and constraint.

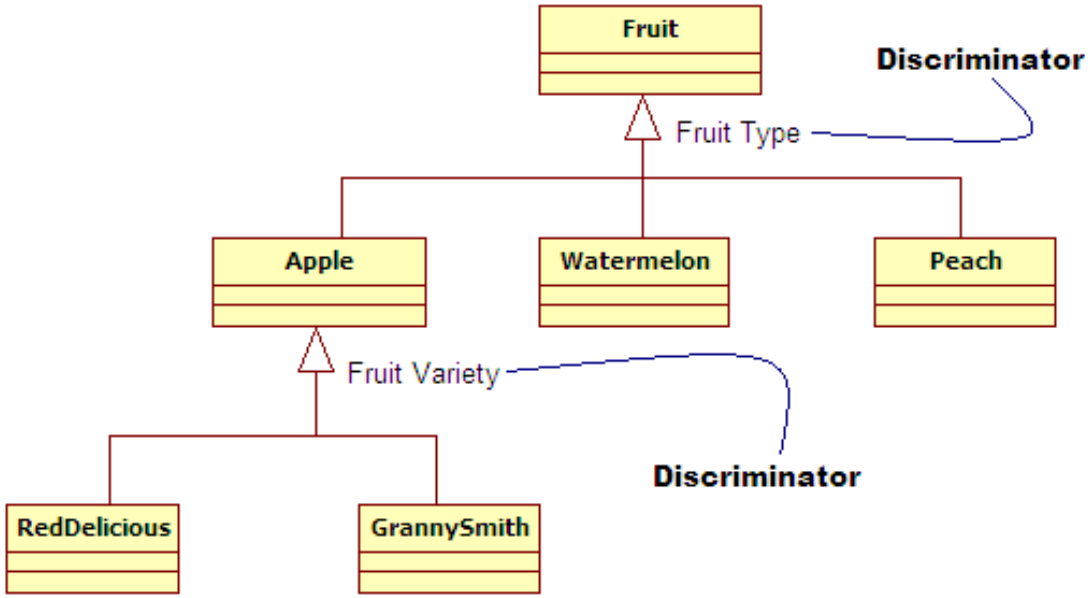


Renesas Design Viet Nam Co., Ltd.

Elements of Generalization

- ***Superclass***
  - Contain some combination of attributes, operations and associations that are common to two or more types of objects that share the same purpose.
- ***Subclass***
  - Contain some combination of attributes, operations and associations that are unique to a type of object that is partially defined by the superclass.
- ***Abstract class***
  - Cannot create object (cannot be instantiated).
  - Any superclass that defines at least one operation that does not have a method.
  - Only a superclass can be abstract.
- ***Concrete class***
  - Class has a method for every operation (can create object).
- ***Discriminator***
  - An attribute or rule describes how to identify the set of subclasses for a superclass.

An example of Discriminator

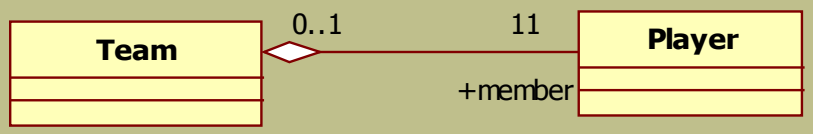


*The superclass **Fruit** is specialized into 3 subclasses using discriminator **Fruit Type***
*The superclass **Apple** is specialized into 2 subclasses using discriminator **Fruit Variety***

Renesas Design Viet Nam Co., Ltd.

Aggregation and Composition

- Aggregation indicates that a class actually owns but may share objects of another class.
- Composition is used to specify the internal parts that make up a class.



Aggregation notation

1 Team is comprised of 11 Players
A Player may exist independent from the Team

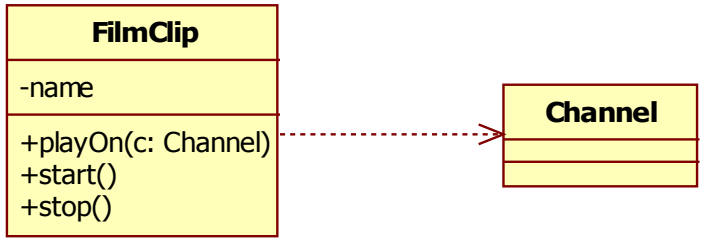
Composition notation

1 Book is composed of 1 or more Chapters
The Chapter would not continue to exist on their own

Renesas Design Viet Nam Co., Ltd.

Dependency

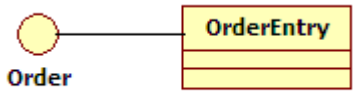
- Dependency is a relationship that states that one class uses the information and services of another class, but not necessarily the reverse.
- Usually, dependencies are used between classes to show that one class uses operations from another class or it uses variables or arguments typed by the other class.
- A dependency can have a name in case that there is a model with many dependencies needed to be distinguished.



Realization

- A realization is a semantic relationship between classes in which one class implements an interface specified by another class.
- Realization is used in two circumstances: in the context of interfaces and in the context of collaborations.

Realization using canonical form

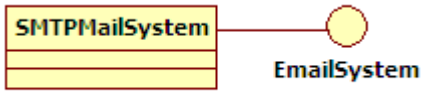


Realization using elided form

Renesas Design Viet Nam Co., Ltd.

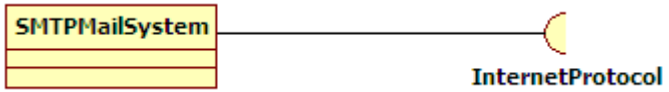
Interface

- An interface is *a collection of operations* that are used to specify a service of a class or a component.
- Interfaces do not specify any implementation.
- An interface may have any number of operations.
- There are 2 kinds of interfaces
  - Provided interface represents services provided by the class.
  - Required interface represents services that the class requires from other classes.



Provided interface

Class SMTPMailSytem is an implementation of the interface EmailSystem



Required interface

Class SMTPMail uses InternetProtocol implemented by another class

Renesas Design Viet Nam Co., Ltd.

Case Study

Problem statement

"One system is designed to inventory and ship uniquely and identified products. These products may be purchased directly from vendors and resold as is, or we can package vendor products together to make our own custom product. Customers place orders for one or more items, but we acknowledge interested customers in the system whether they have purchased yet or not. Each item corresponds to a product. We identify each product using a unique serial number. The Customer may inquire on the status of his Orders using order number.

Shipment of products from vendors is received and placed into inventory. Each product is assigned to a location so that we can easily find it later when filling orders. Each location has a unique location identifier. Customer orders are shipped as the products become available, so there may be more than one shipment to satisfy a single customer order. But a single shipment may contain products from multiple orders. Any items that have not been shipped are placed on a backorder with a reference to the original order."

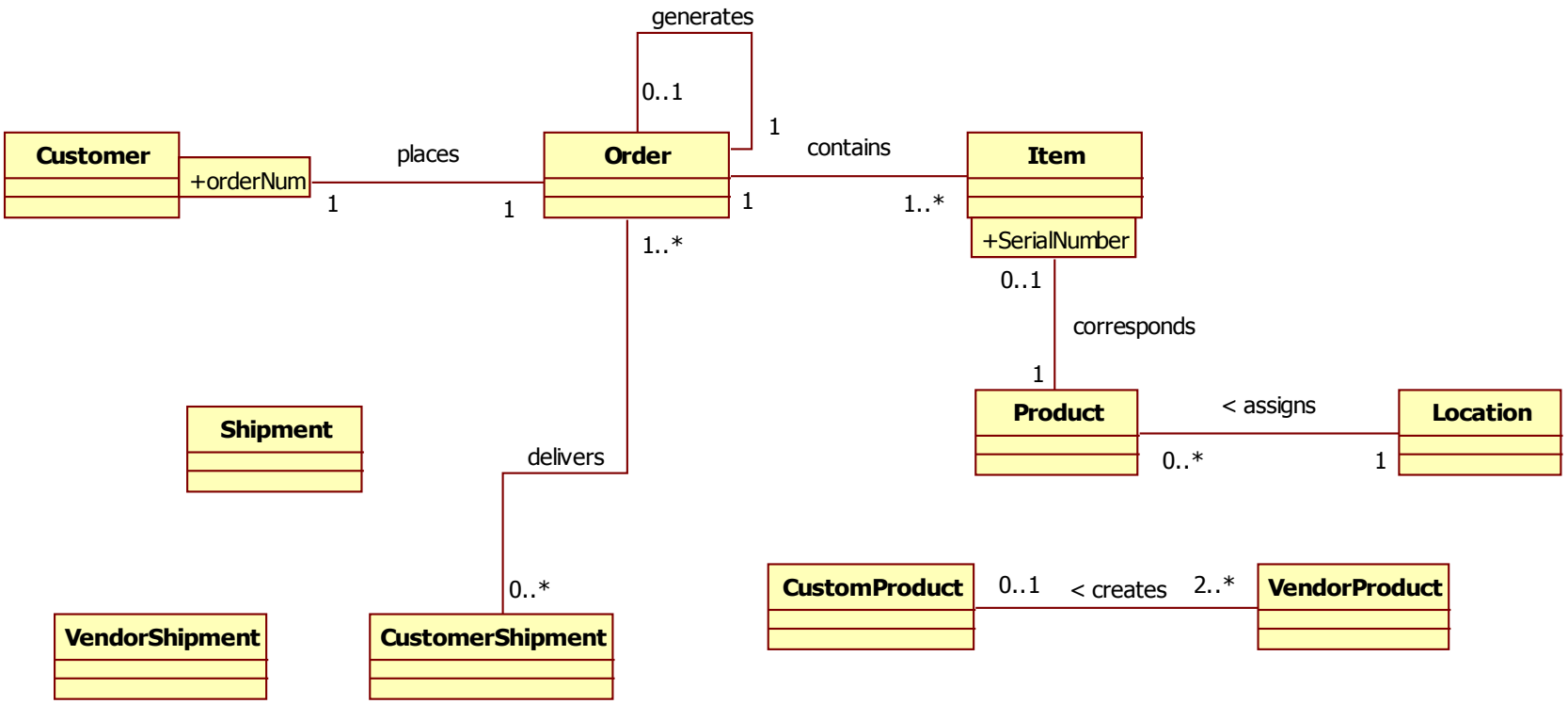
Steps to build a class diagram

- Identify the classes, name and define them.
- Identify, name and define the associations between pairs of classes (including reflexive associations).
- Assign multiplicity and constraints.
- If naming an association is difficult, try role names.
- Evaluate each association to determine whether it should be defined as aggregation. If it is aggregation, then could it be composition?
- Evaluate the classes for possible specialization or generalization.

Build the Class Diagram

- ***Identify the classes***
  - Customer, Order, Item, Product, CustomProduct, VendorProduct, Location, Shipment, VendorShipment, CustomerShipment
- ***Identify the association between pairs of classes***
  - Customer *places* Order.
  - Order *contains* Item.
  - Each Item *corresponds* to a Product by a *serial number*.
  - An Order that is not filled completely will *generate* another Order that it refers to as a backorder and that backorder is associated with the Order that *generated* it.
  - The Order is *delivered* to the Customer via a Shipment.
  - Each Product is *assigned* to a Location.
  - CustomProducts can be *created* by packaging VendorProducts.

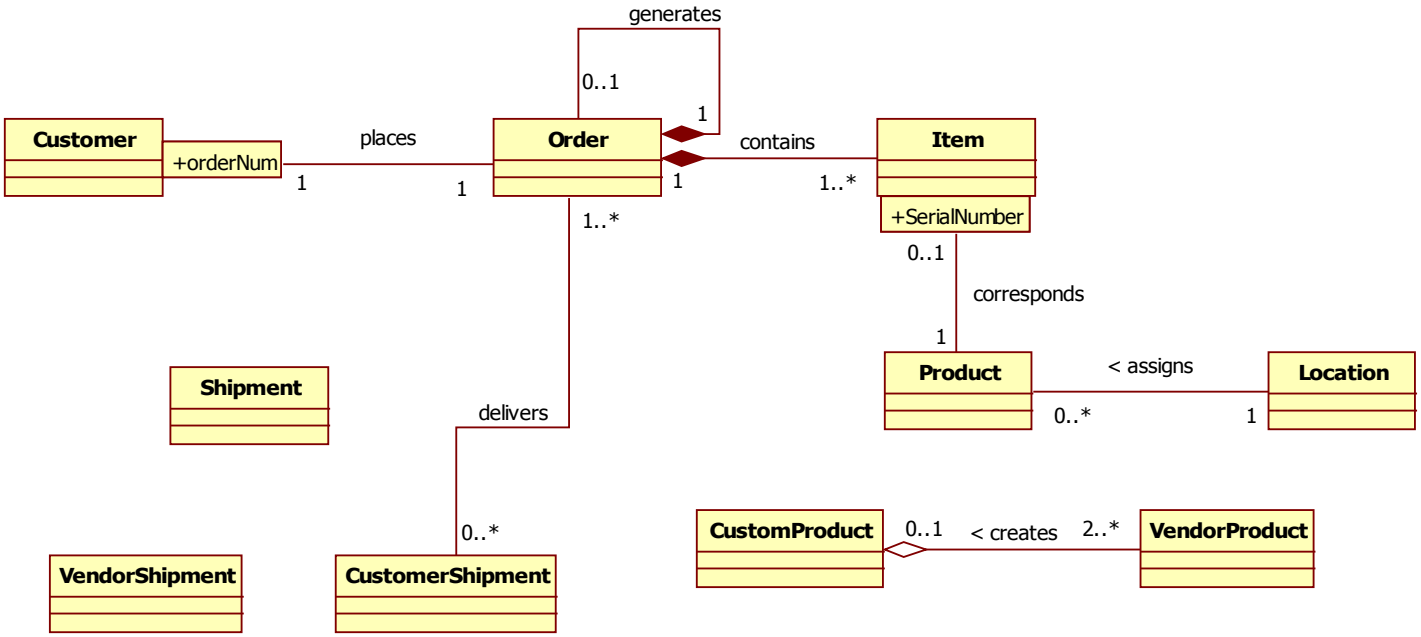
Identify Classes and Associations – *ILLUSTRATION*



Renesas Design Viet Nam Co., Ltd.

Identify Aggregation and Composition

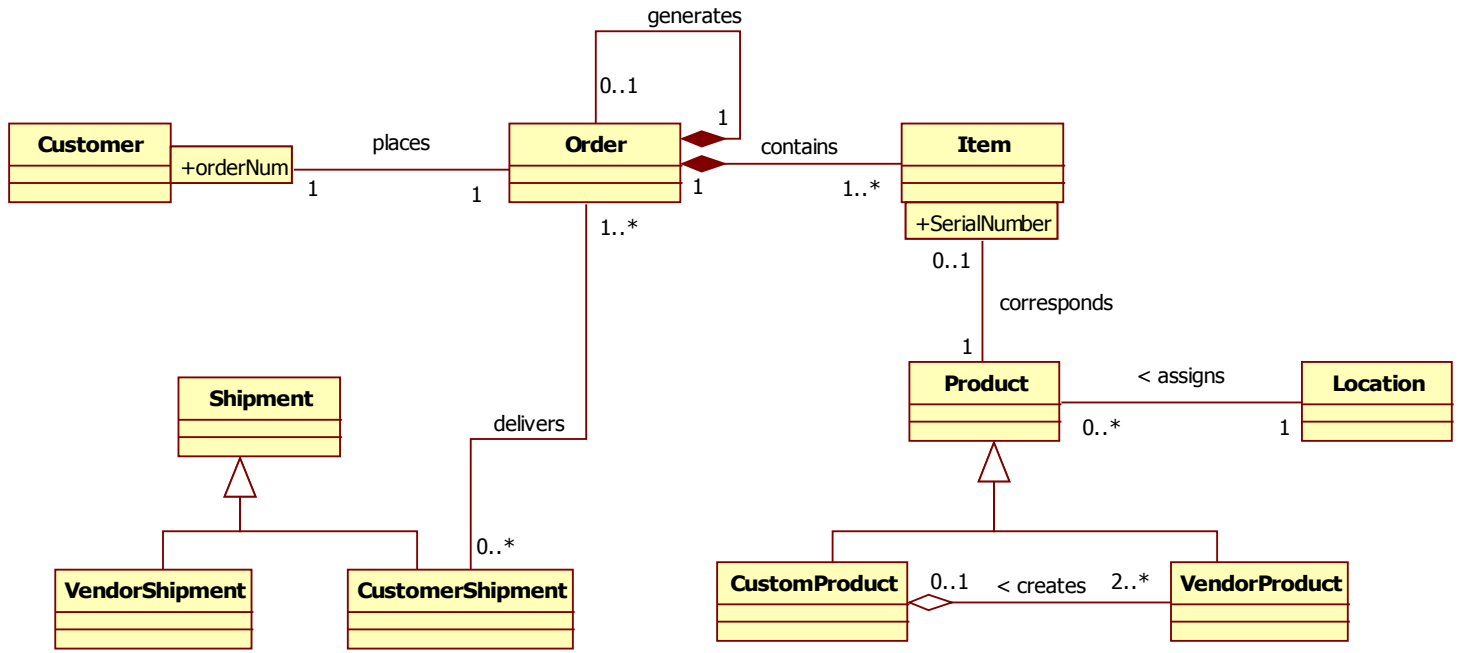
- Each backorder is generated by an Order → composition.
- An Order is constructed using one or more Items → composition.
- CustomProducts can be created using VendorProducts, but either of them can exist on their own → aggregation.



Renesas Design Viet Nam Co., Ltd.

Identify Generalization

- There are 2 kinds of Shipment: VendorShipment and CustomerShipment.
- Products are divided into 2 types: VendorProducts delivered by vendor, and CustomProducts created by packaging VendorProducts.



Renesas Design Viet Nam Co., Ltd.

2. USE CASE DIAGRAM

Overview

- Use case diagram models the functionality of the system as perceived by outside users.
- ***Purposes***
  - Use case diagram is intended to *list the actors and use cases* and show which actors participate in each use case.
  - Use case diagram is important for visualizing, specifying and documenting the behavior of an element.
  - Use case diagram is also important for testing executable systems through forward engineering and for comprehending executable systems through reverse engineering.

A Use Case

- A use case is a coherent unit of externally visible functionality provided by a system unit and expressed by sequences of messages exchanged by the system unit and one or more actors of the system unit.
- The purpose of a use case is to define a piece of coherent behavior without revealing the internal structure of the system.
- The dynamics of a use case may be specified by UML interactions, shown as state chart diagrams, sequence diagrams, collaboration diagrams or informal text descriptions.
- A use case is like a system operation, an operation invocable by an outside user.
- A use case is a logical description of a slice of system functionality.
- A class can have more than one use case. Each use case must be mapped onto the classes that implement a system.

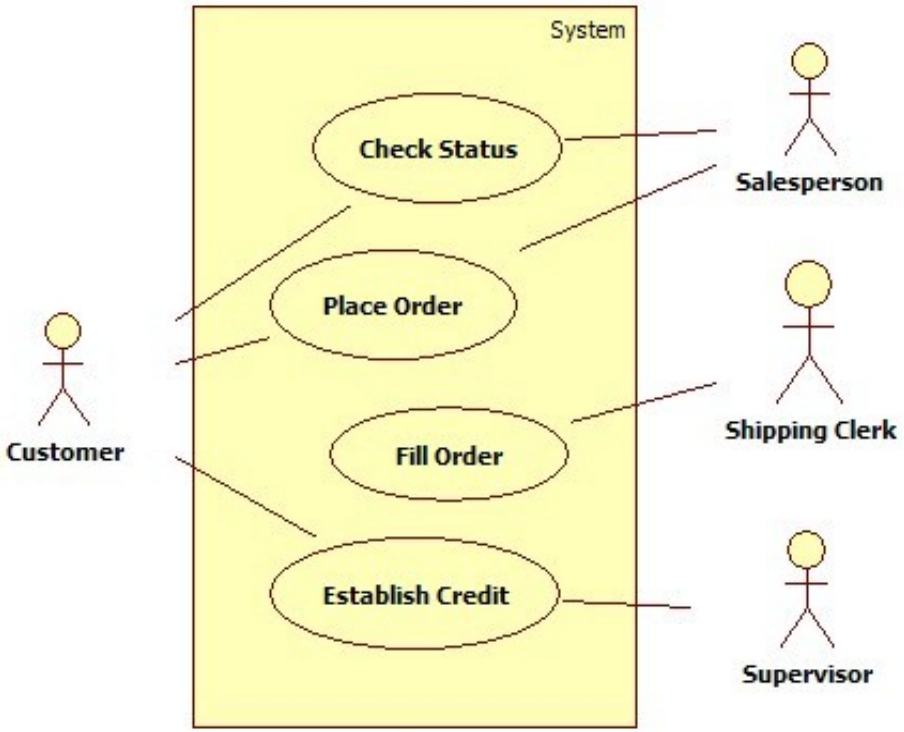
Actor

- An actor represents a coherent set of roles that users of use cases play when interacting with these use cases.
- An actor represents a role that a human, a human device or even another system plays with a system.
- Each actor participates in one or more use cases.
- Two kinds of actors
  - *Primary actor* is the one for whom the use case produces an observable result.
  - *Secondary actors* constitute the other participants of the use case. Secondary actors are requested for additional information; they can only consult or inform the system when the use case is being executed.
- Actors can be identified in context diagram.
  - *Context diagram* is a class diagram in which each actor is linked to a central class representing the system by an association, which enables the number of instances of actors connected to the system at a given time to be specified.

Renesas Design Viet Nam Co., Ltd.

Association

- The communication path between an actor and a use case that participates in.
- This example illustrates the idea of actor, use case and association.

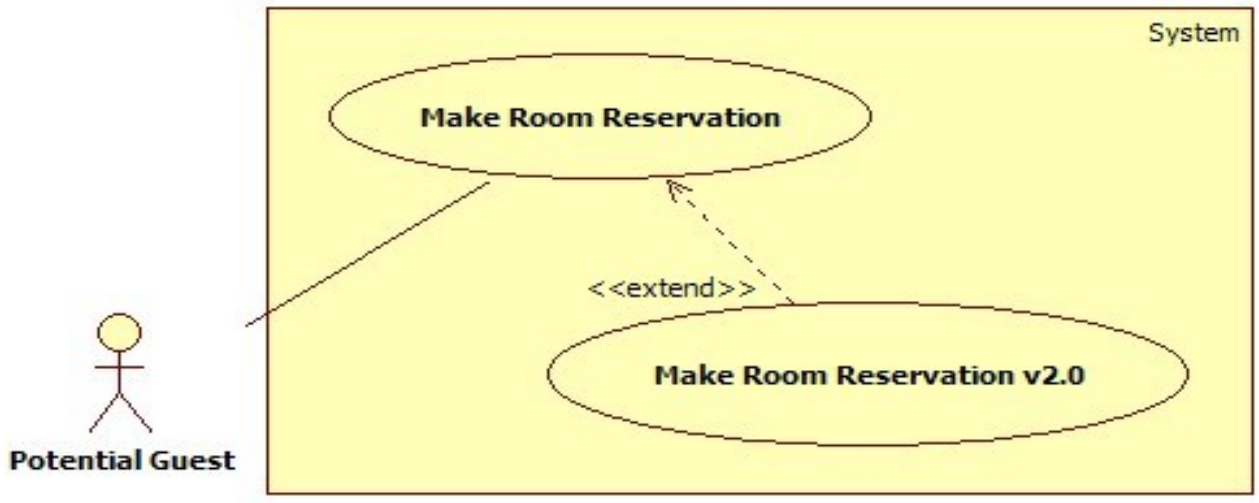


Renesas Design Viet Nam Co., Ltd.

Extend

- The insertion of additional behavior into a base use case that does not know about it.
- Three reasons for extension
  - ***Changed capability***

    Extension can be useful with later release when you have changed a use case significantly.

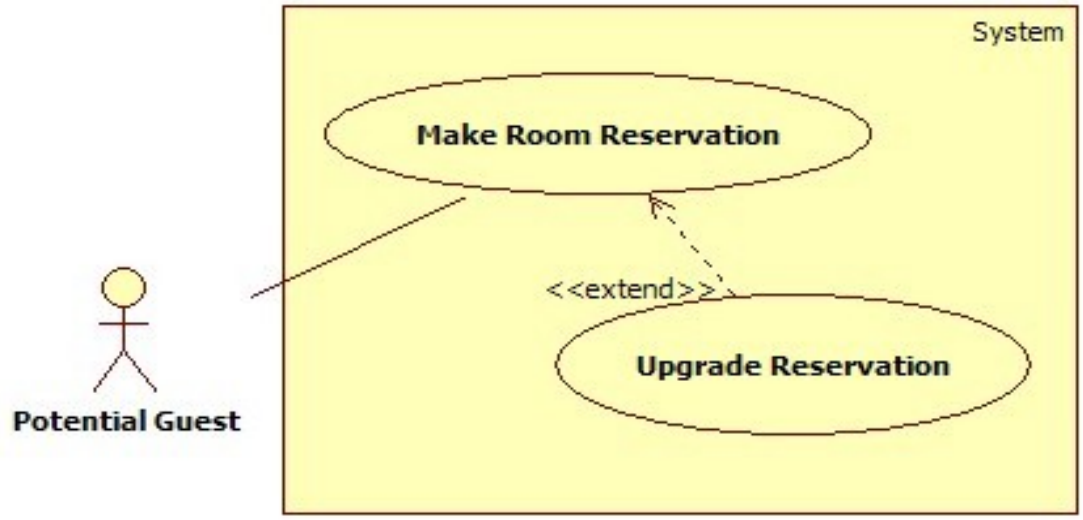


Renesas Design Viet Nam Co., Ltd.

An example of Extend (1)

- ***Major variation***

If you have a major alternative path in the use case and it's complex enough to have its own alternative paths.

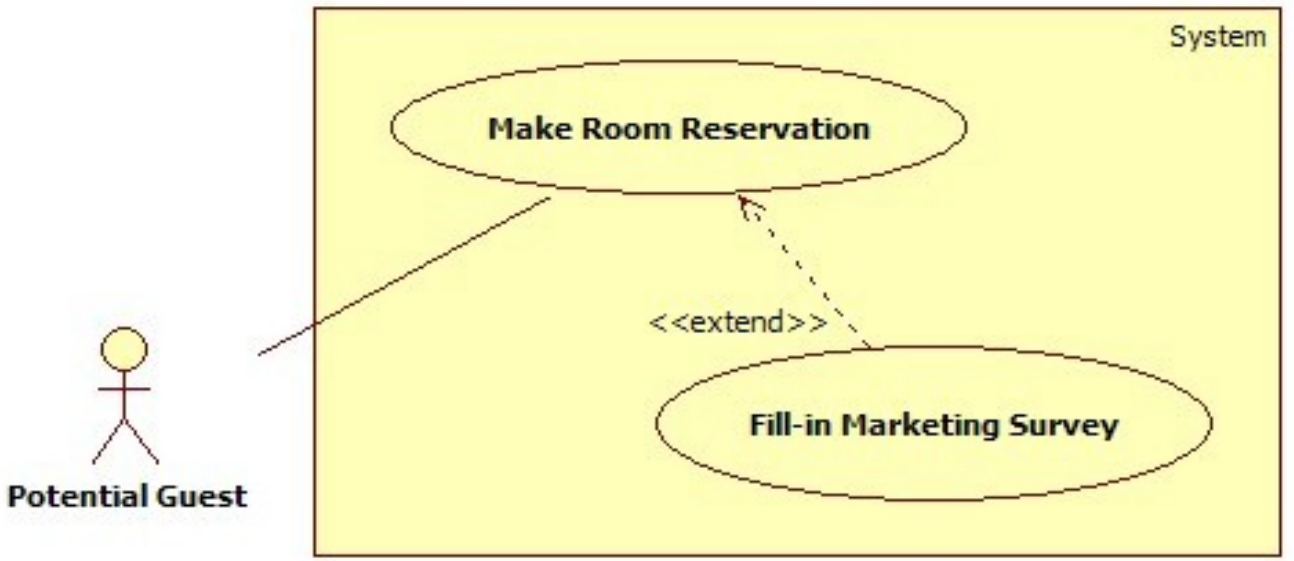


Renesas Design Viet Nam Co., Ltd.

An example of Extend (2)

- ***Optional subgoal***

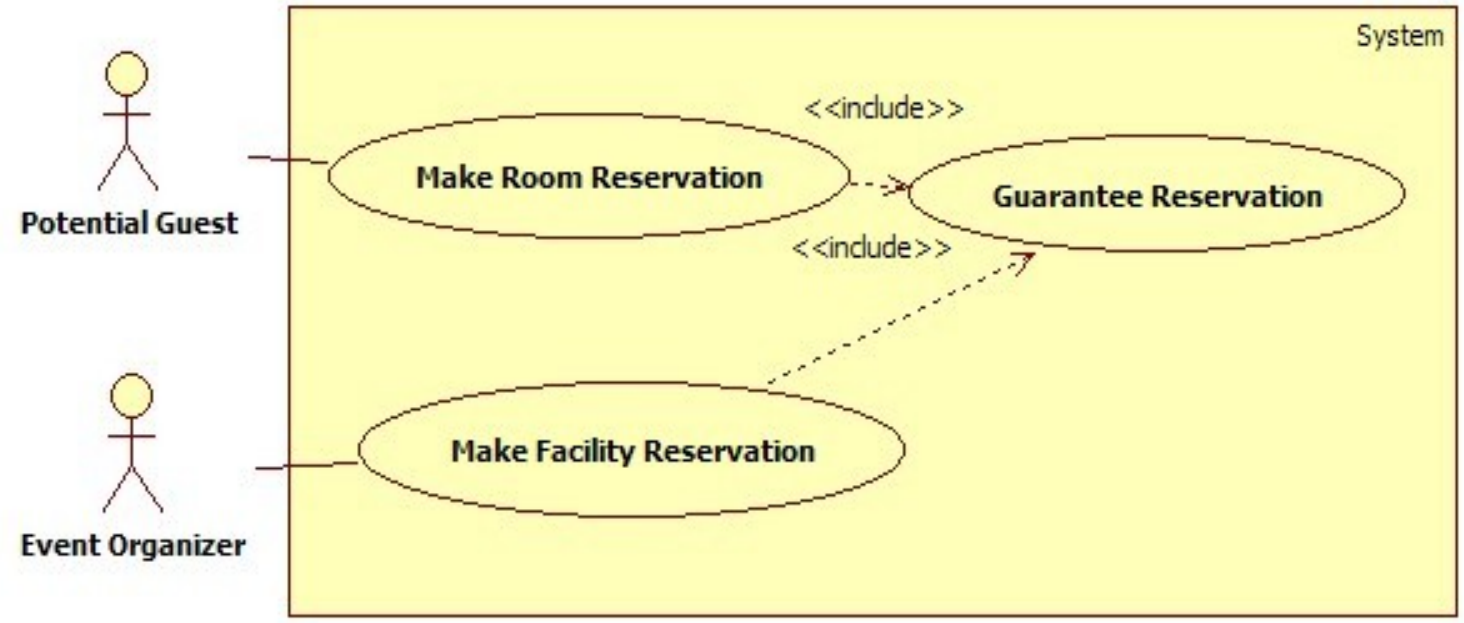
If you have parts of the use case that would be optional to implement to meet the actor's goals.



Renesas Design Viet Nam Co., Ltd.

Include

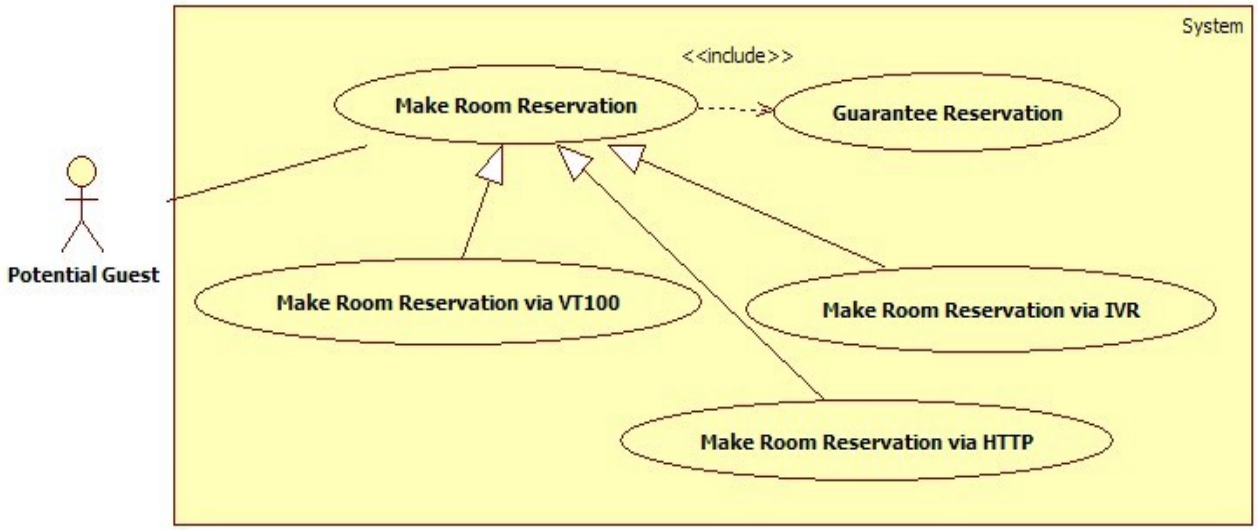
- The insertion of additional behavior into a base use case that explicitly describes that insertion.
- The new use case is not special case of the original use case and cannot be substitute it.



Renesas Design Viet Nam Co., Ltd.

Use Case Generalization (1)

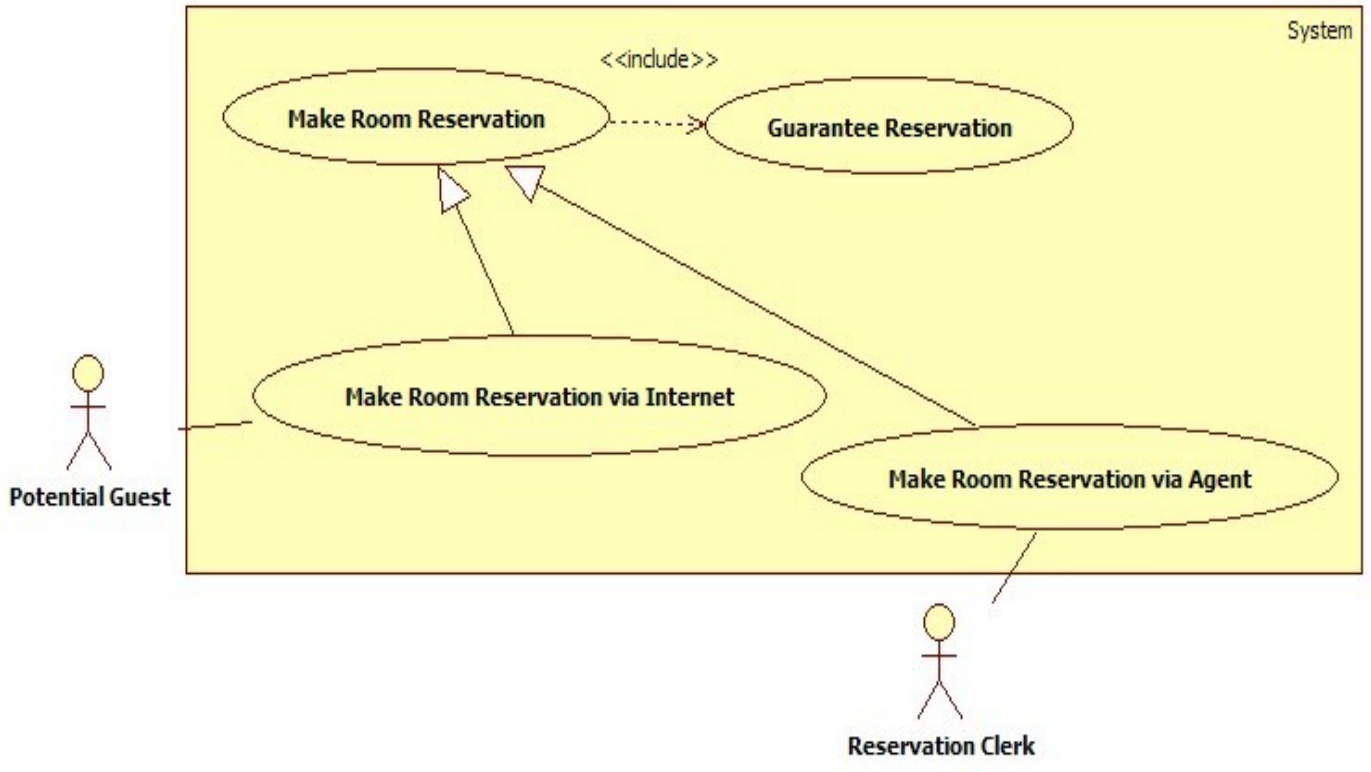
- A relationship between a general use case and a more specific use case that inherits and adds features to it.
- Two common circumstances to generalize use cases.
  - ***Differing mechanisms for the same goal***
    - More than one alternative technique or approach that the system uses to help the actors get their goals accomplished.
    - Approaches will still be sharing a bit: requirements, business rules and data validation.



Renesas Design Viet Nam Co., Ltd.

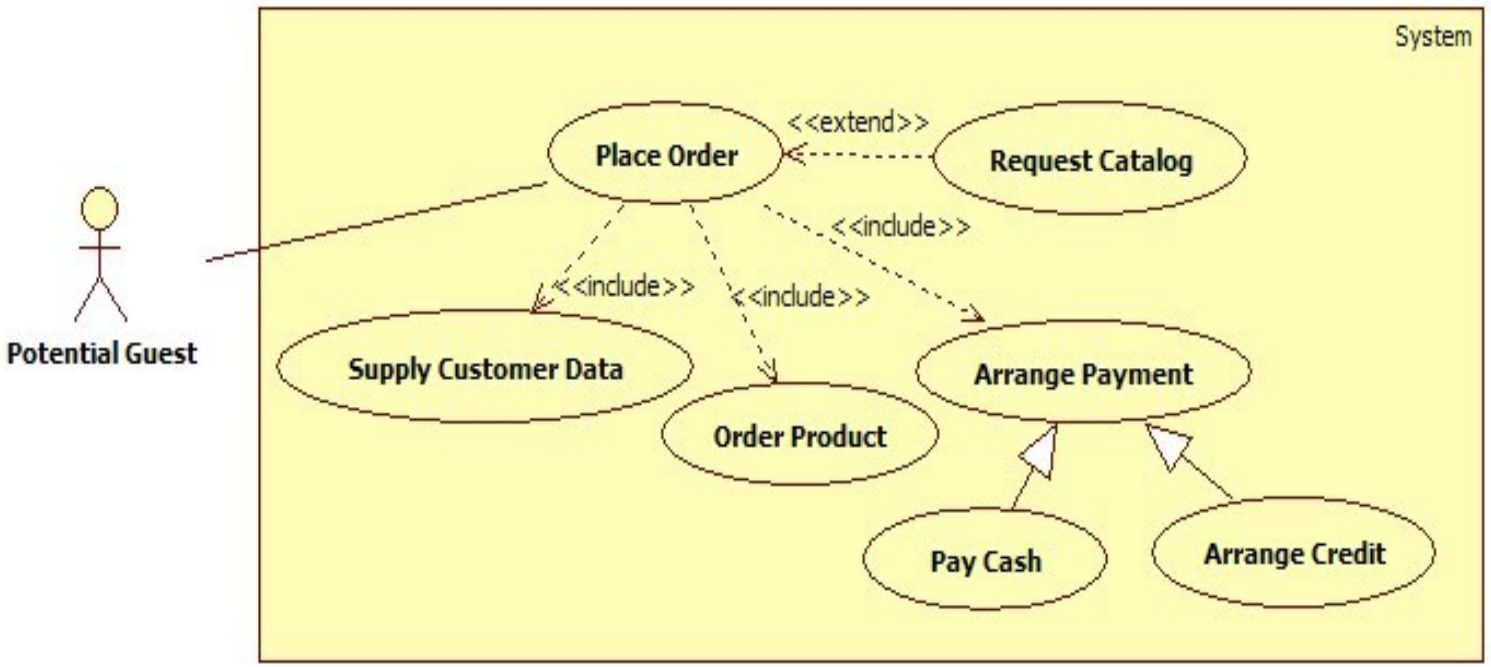
Use Case Generalization (2)

- ***Differing agents for the same goal***
  - More than 1 actor trying to accomplish the same goal.
  - Actors have separate privileges, capabilities or user interfaces.



Renesas Design Viet Nam Co., Ltd.

An example of Use Case Relationship



Renesas Design Viet Nam Co., Ltd.

Use Case Description

- Use case descriptions are simply less formal descriptions of use cases that contain all information necessary to create a formal use case diagram.
  - ***Purpose:*** Used to explain the dynamics of a use case in detail.
  - This description contains a list of all the interactions between the actors and the system.
  - *Main parts*: main flow, alternative flows, error flows
    - *Main flow* (also known as basic flow of events or normal path): is the main success scenario of the use case
    - *Alternative flow:* the primary actor achieves his goal
    - *Error flow:* the actor's goal is not achieved and the use case fails.
  - Although it is possible to skip the use case description step and move directly into creating use case diagrams and other diagrams, users often have difficulty describing their business processes using use case diagram without creating use case description first.
- Scenario represents a particular succession of sequences, which is run from beginning to end of the use case.

Renesas Design Viet Nam Co., Ltd.

Guidelines of Usage – How to construct?

- Identify actors and context diagram
- Identify use cases
- Create use case diagrams
- Textual description of use cases
- Organize use cases
- ***Case study***

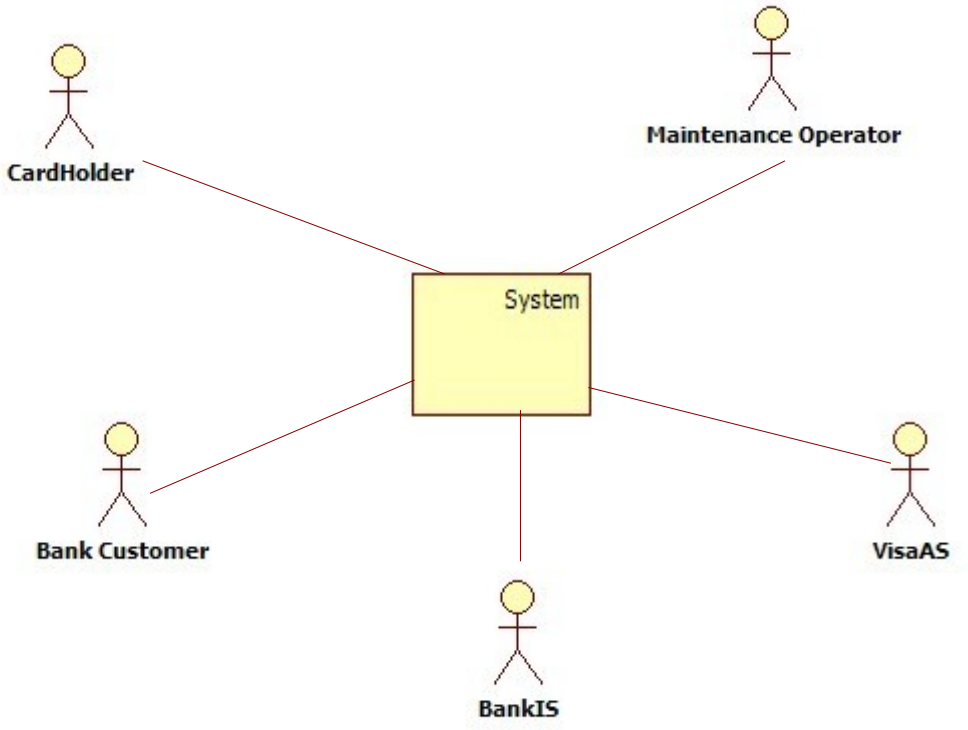
This case study concerns a simplified system of the automatic teller machine (ATM). The ATM offers the following services
1. Distribution of money to every holder of a smart card via a card reader and a cash dispenser
2. Consultation of account balance, cash and cheque deposit facilities for bank customers who hold a smart card from their bank

Do not forget that
1. All transactions are made secure
2. It is sometimes necessary to refill the dispenser

Renesas Design Viet Nam Co., Ltd.

Case Study (1)

- ***Identify actors and context diagram***
  - ***Primary actors***: CardHolder, Bank customer, Maintenance operator
  - ***Secondary actors***: Visa Authorisation System (Visa AS) and Bank Information System (Bank IS)



Renesas Design Viet Nam Co., Ltd.

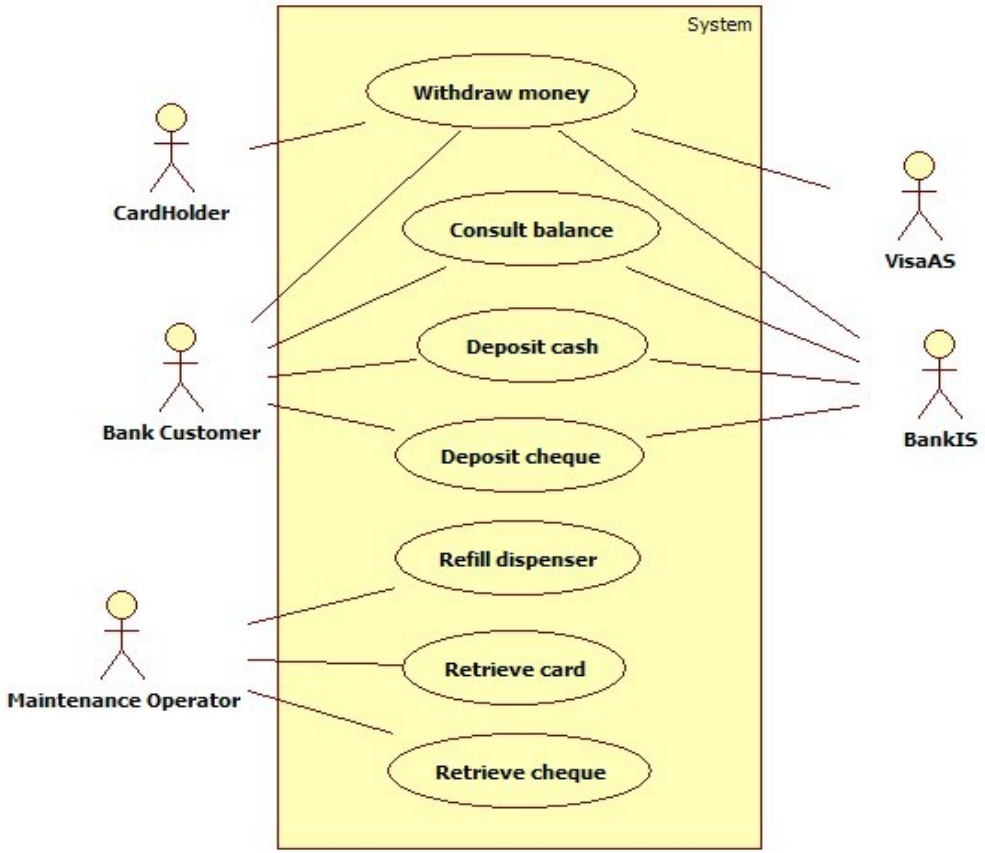
Case Study (2)

- ***Identify use cases***
  - ***CardHolder***
    - Withdraw money
  - ***Bank Customer***
    - Withdraw money
    - Consult the balance of one or more accounts
    - Deposit cash
    - Deposit cheques
  - ***Maintenance Operator***
    - Refill dispenser
    - Retrieve cards that has been swallowed
    - Retrieve cheques that have been deposited
  - ***VisaAS***
    - None
  - ***BankIS***
    - None

Renesas Design Viet Nam Co., Ltd.

Case Study (3)

- Create use case diagrams



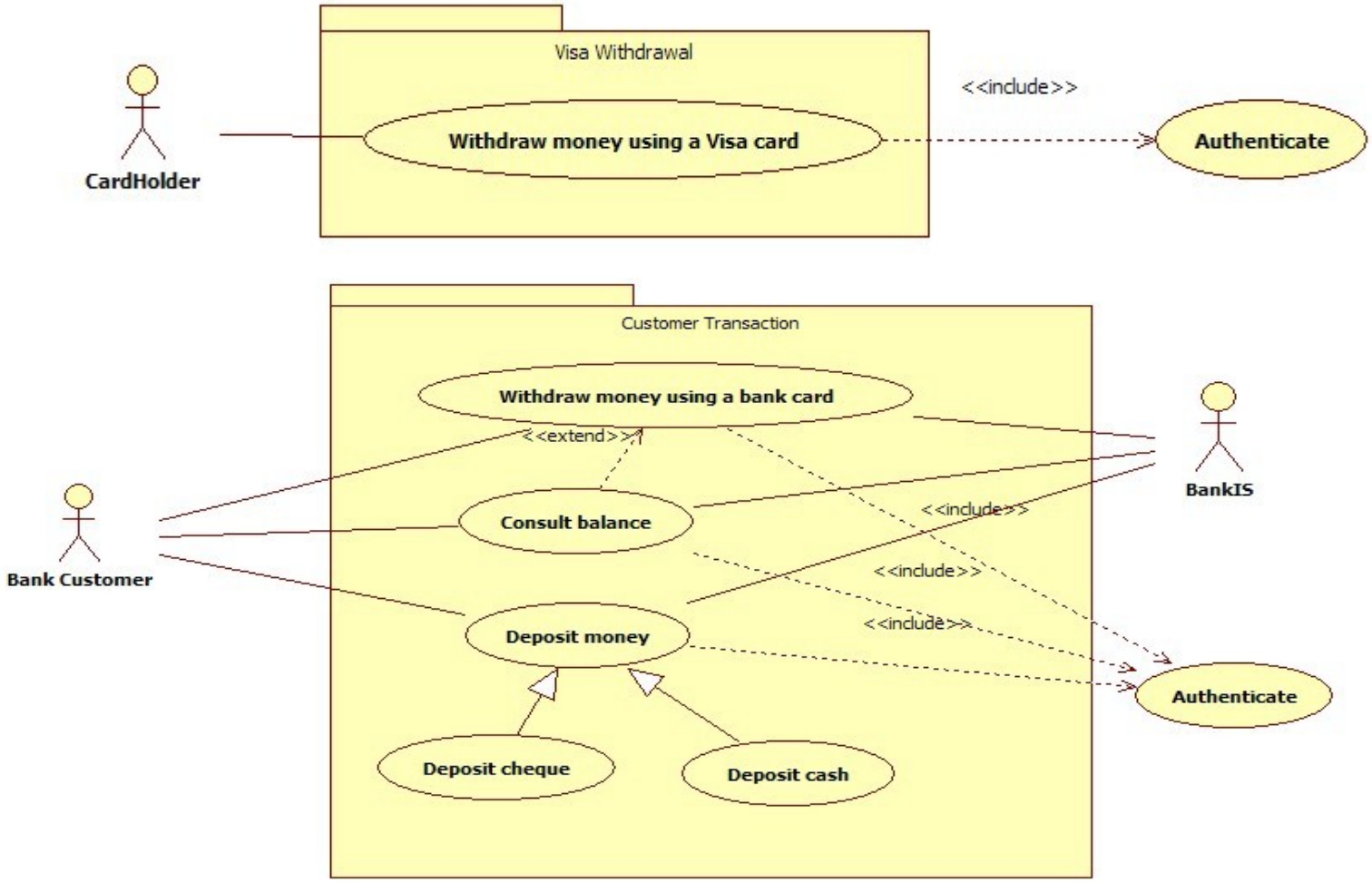
Renesas Design Viet Nam Co., Ltd.

Case Study (4)

- Textual description of use cases.
- Organize use cases
  - Use cases are organized in 2 different and complementary ways
    - By adding include, extend and generalization relationships between use cases.
    - By grouping them into packages to define functional blocks of highest level.

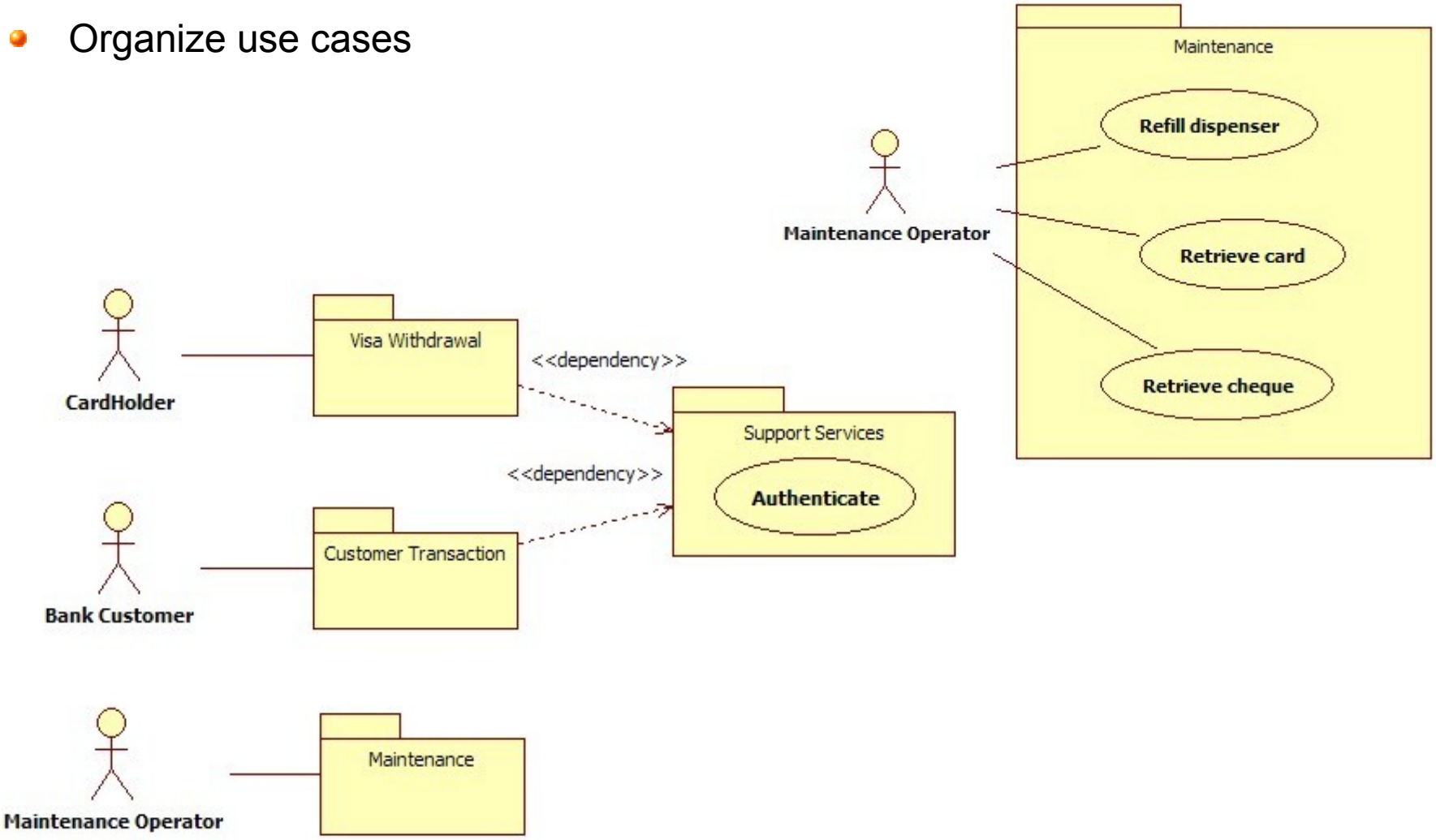
Case Study (5)

- Organize use cases



Case study (6)

- Organize use cases



Renesas Design Viet Nam Co., Ltd.

Thank you for your attention !

Renesas Design Viet Nam Co., Ltd.