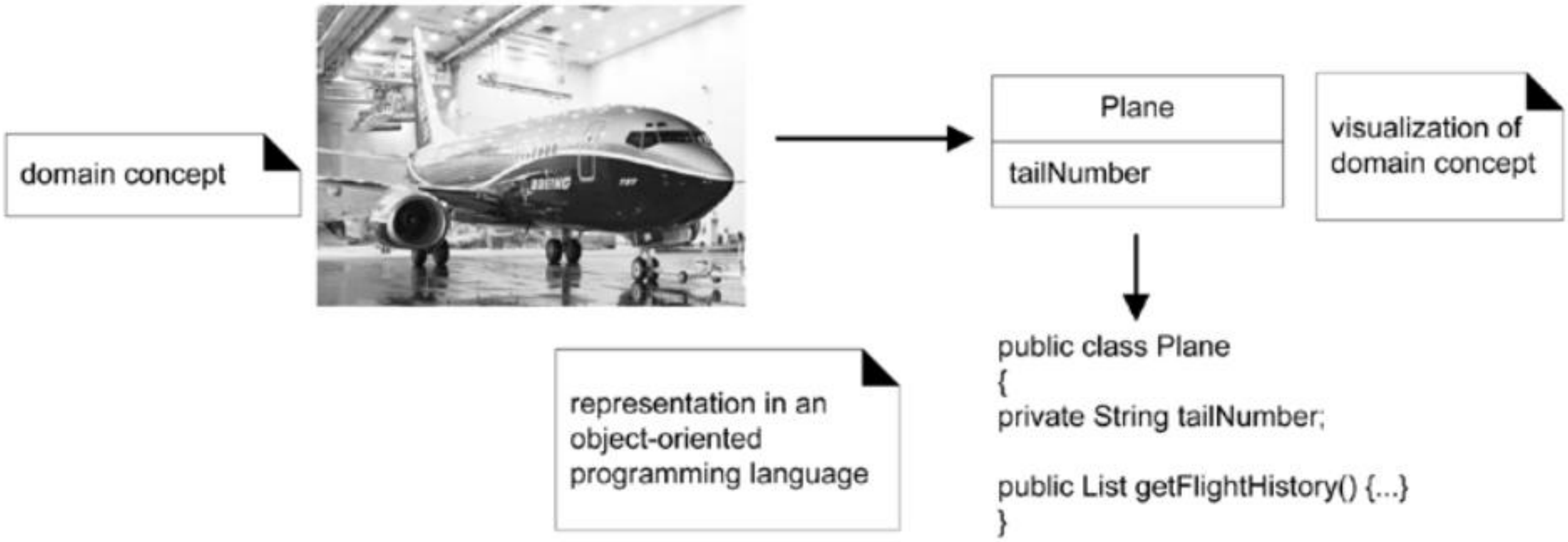
2. UML for OOAD

Object-orientation emphasizes representation of objects



2.1 UML Background

"The Unified Modelling Language (UML) is a graphical language for visualizing, specifying, constructing, and documenting the artifacts of a software-intensive system. The UML offers a standard way to write a systems blueprints, including conceptual things like business processes and system functions as well as concrete things such as programming language statements, database schemas, and reusable software components."

Grady Booch, Ivar Jacobsen, Jim Rumbaugh
Rational Software

[OMG Unified Modelling Language Specification, Version 1.3, March 2000]

2.1 Brief UML History

- Around 1980
  - first OO modelling languages
  - other techniques, e.g. SA/SD
- Around 1990
  - "OO method wars"
  - many modelling languages
- End of 90's
  - UML appears as combination of best practices

2.1 Why UML?

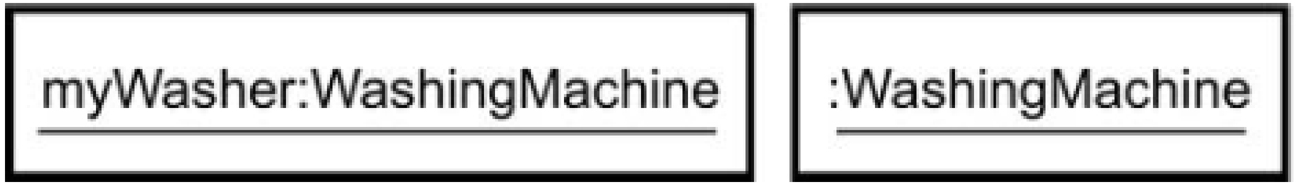
- We need a common language
  - discuss software systems at a black- (white-) board
  - document software systems
  - UML is an important part of that language
  - UML provides the "words and grammar"

2.2 Classes in UML

- Classes describe objects
  - Interface (member function signature)
  - Behaviour (member function implementation)
  - State bookkeeping (values of data members)
  - Creation and destruction
- Objects described by classes collaborate
  - Class relations $\rightarrow$ object relations
  - Dependencies between classes

2.2 UML Summary

Figure 1.2. Two UML object icons—The icon on the left represents a named object, the icon on the right represents an anonymous object.



2.2 UML Summary

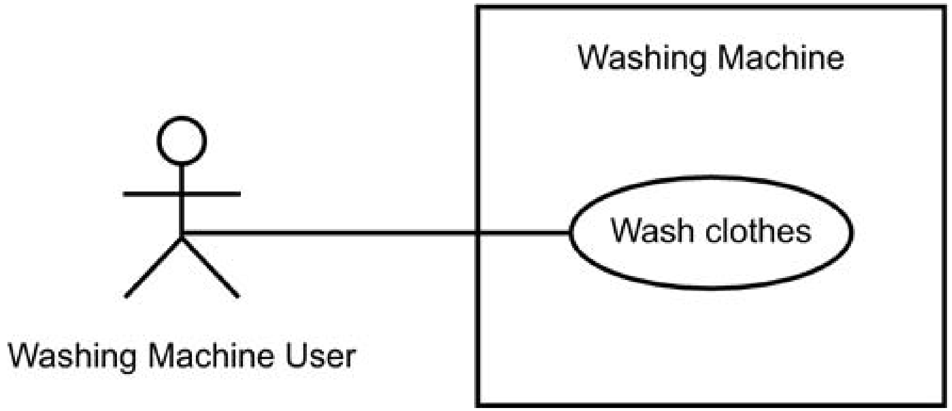


Figure 1.3. The UML use case diagram.

2.2 UML Summary

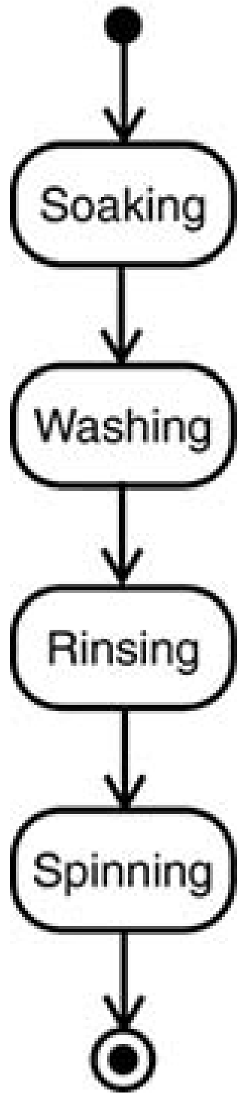


Figure 1.4. The UML state diagram.

2.2 UML Summary

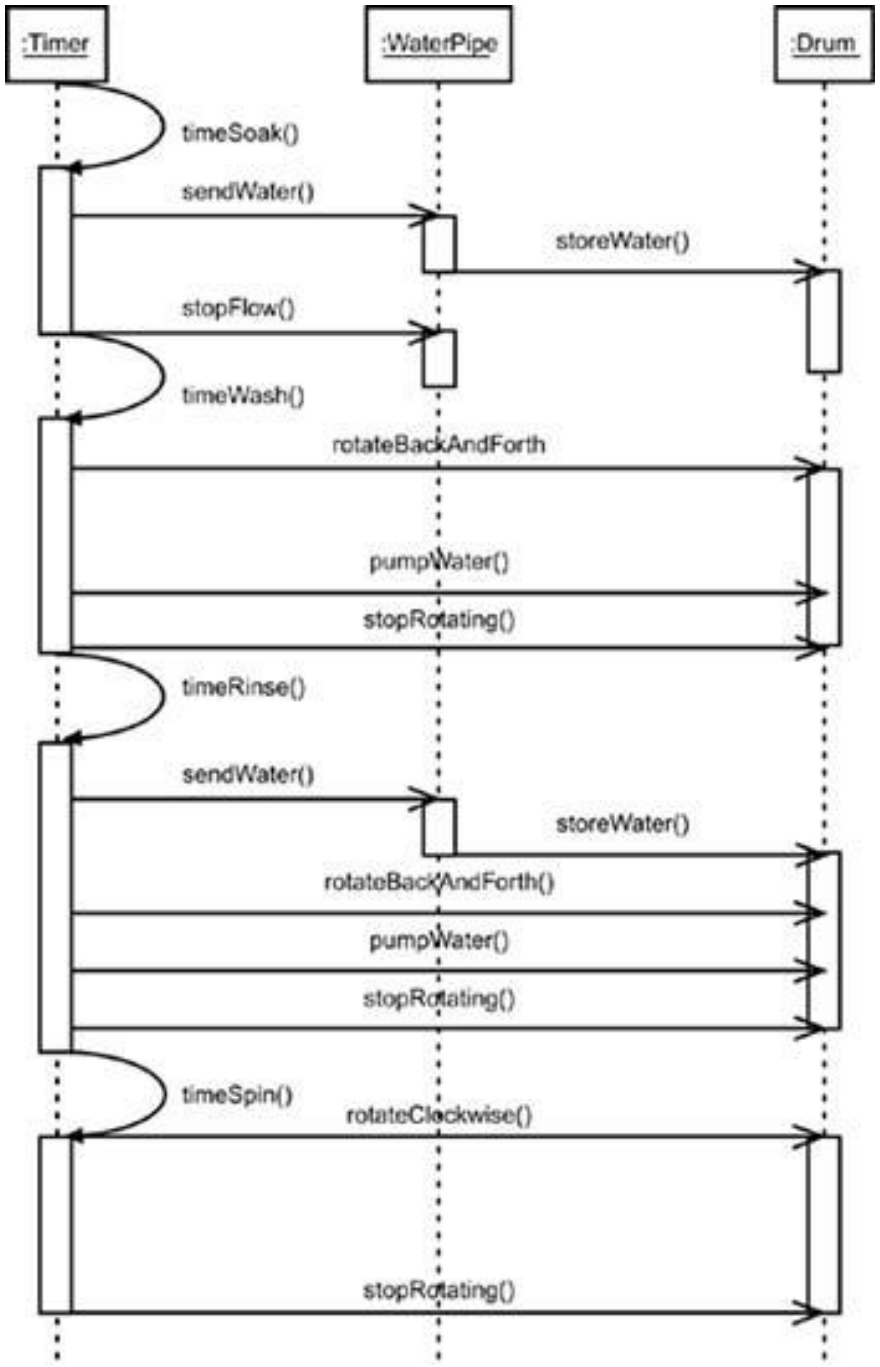


Figure 1.5. The UML sequence diagram.

2.2 UML Summary

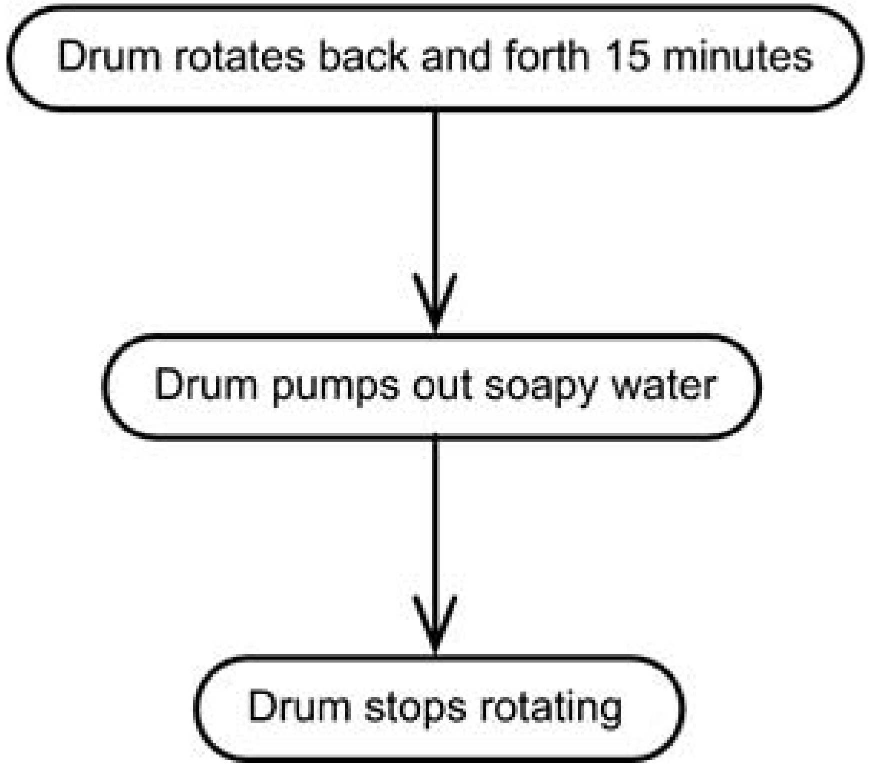


Figure 1.6. The UML activity diagram.

2.2 UML Summary

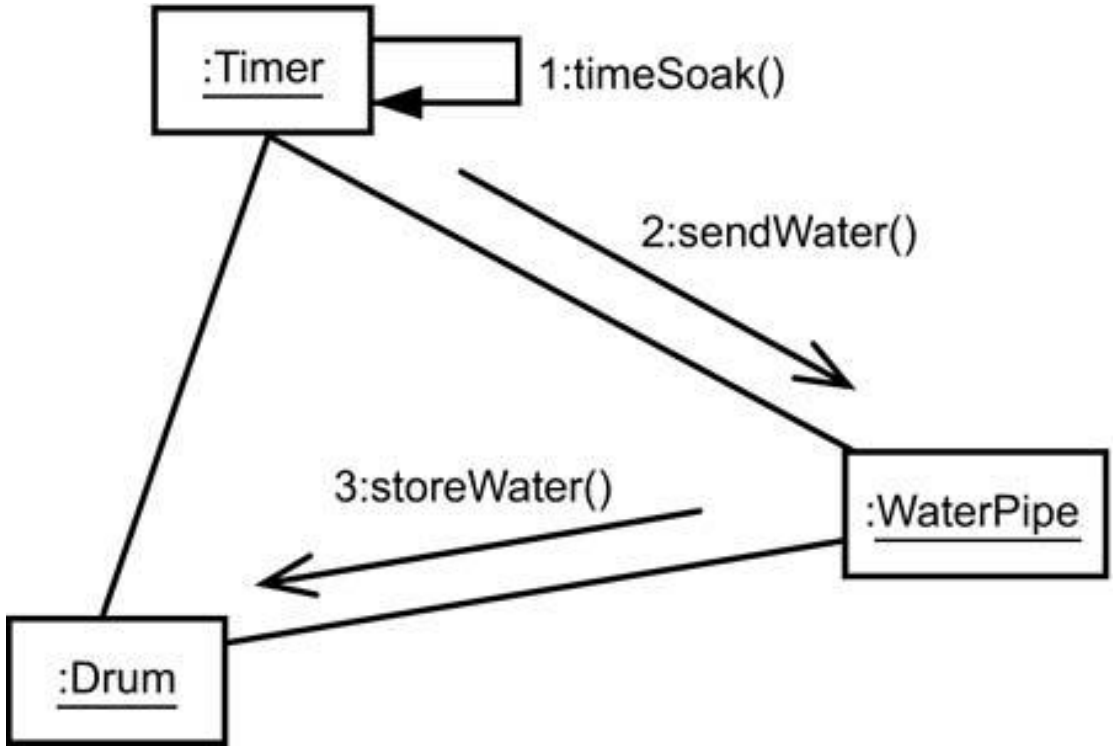


Figure 1.7. The UML communication diagram.

2.2 UML Summary

Figure 1.8. The software component icon in UML 1.x.

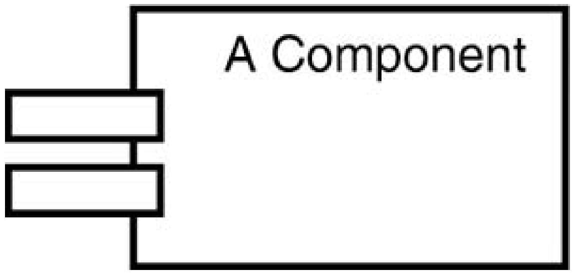
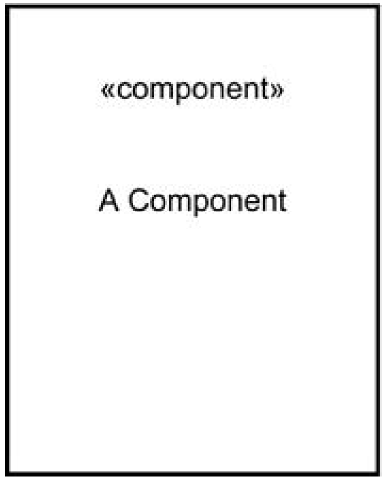
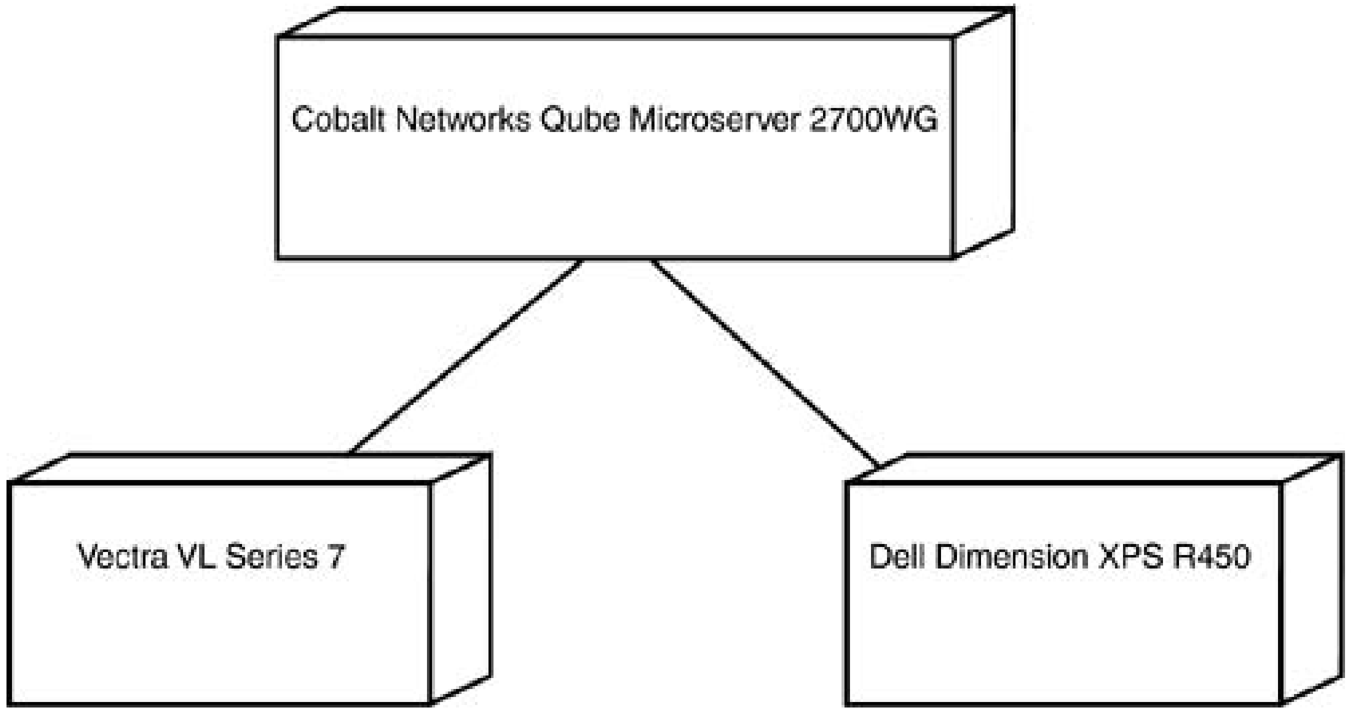


Figure 1.9. The software component icon in UML 2.0.



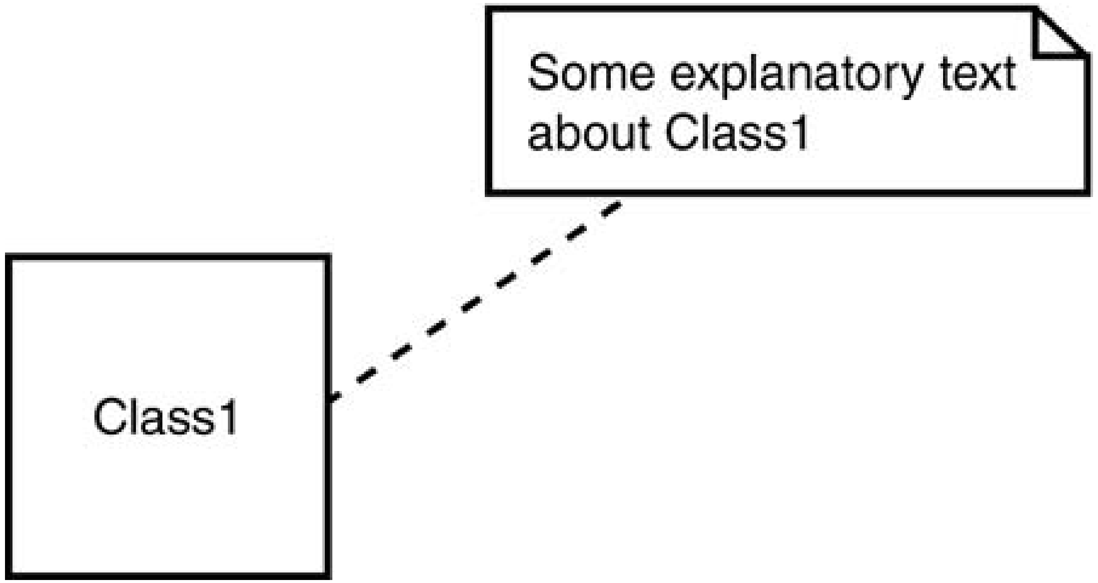
2.2 UML Summary

Figure 1.10. The UML deployment diagram.



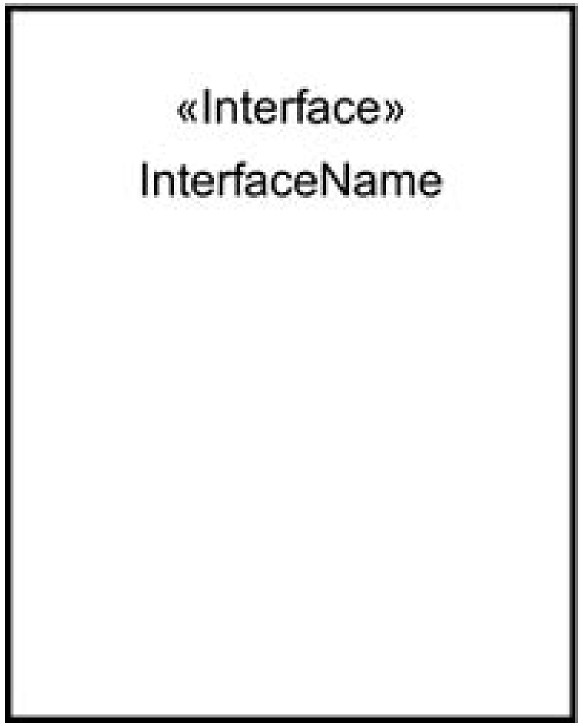
2.2 UML Summary

Figure 1.11. In any diagram you can add explanatory comments by attaching a note.

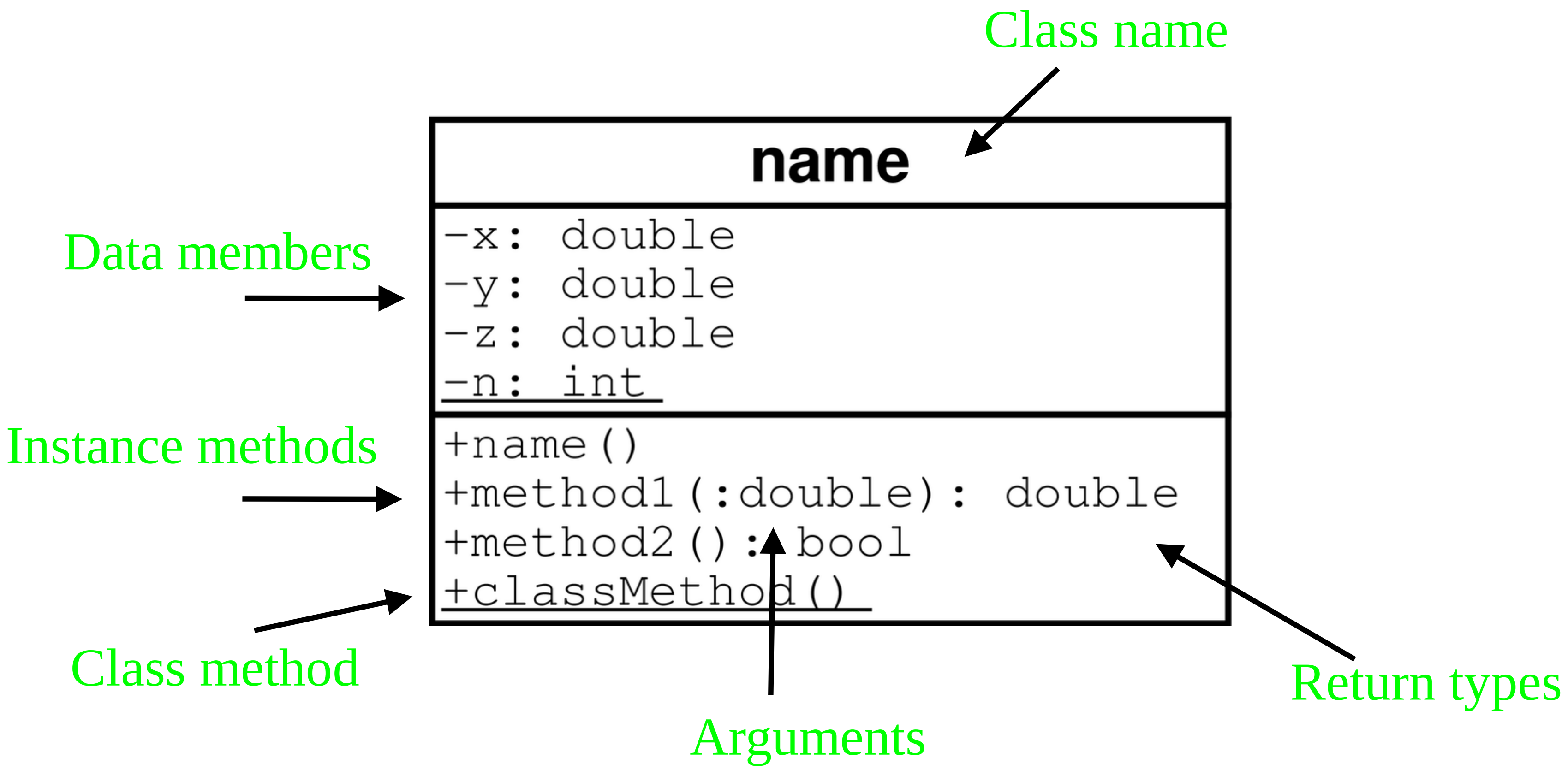


2.2 UML Summary

A stereotype is an existing UML element with the addition of a keyword in guillemets. The keyword indicates that the element is used in a somewhat different way than originally intended.



2.2 UML Class



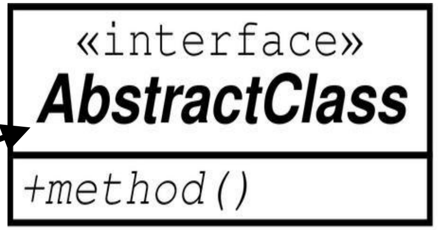
Data members, arguments and methods are specified by
visibility name : type

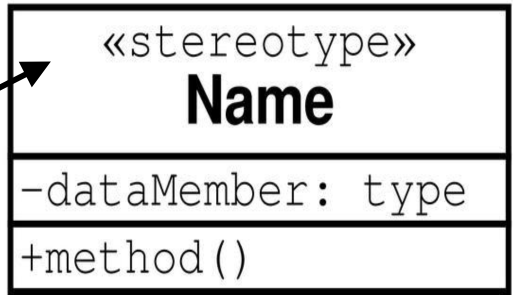
2.2 Class Name

The top compartment contains the class name

Abstract classes have italicised names
Abstract methods also have italicised names

Stereotypes are used to identify groups of classes, e.g interfaces or persistent (storeable) classes





2.2 Class Attributes

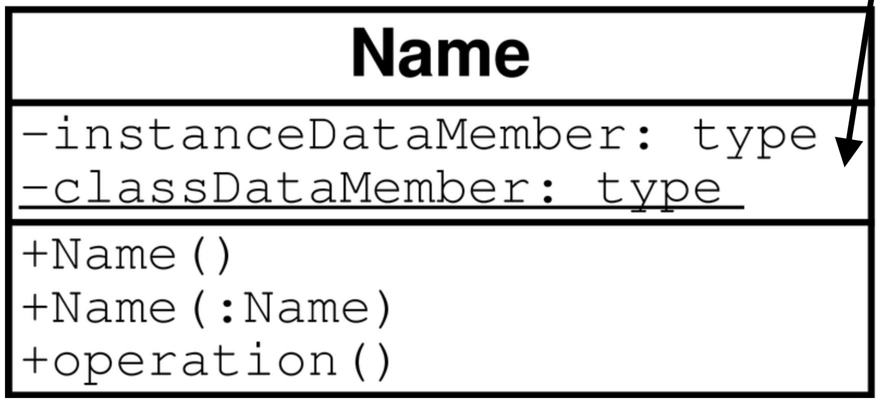
Attributes are the instance
and class data members

Class data members (underlined)
are shared between all instances
(objects) of a given class

Data types shown after ":"

Visibility shown as
- + public
- - private
- # protected

Attribute
compartment



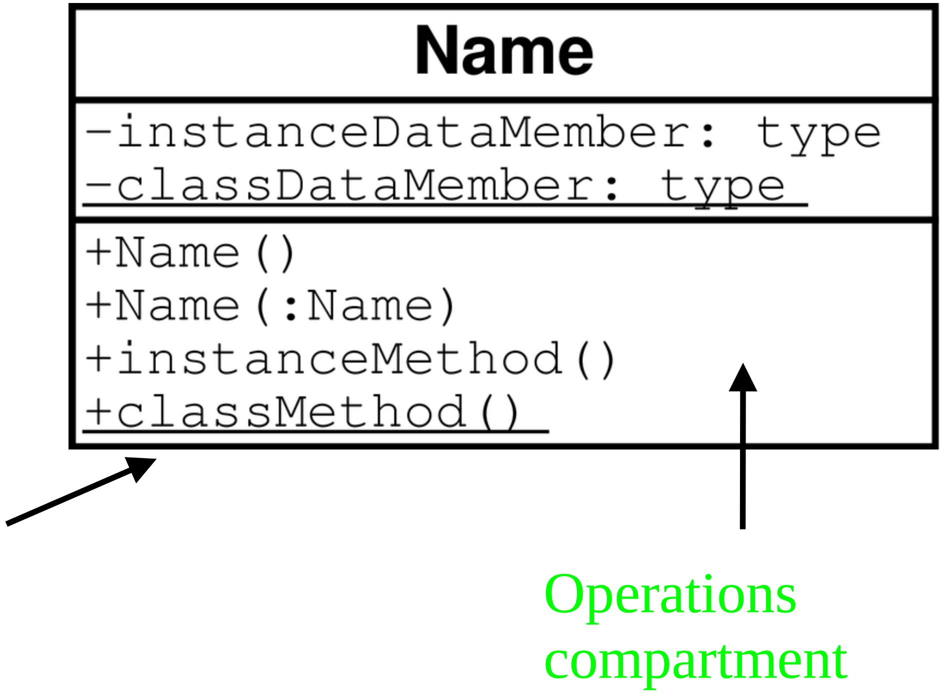
visibility name : type

2.2 Class Operations (Interface)

Operations are the class methods with their argument and return types

Public (+) operations define the class interface

Class methods (underlined) have only access to class data members, no need for a class instance (object)



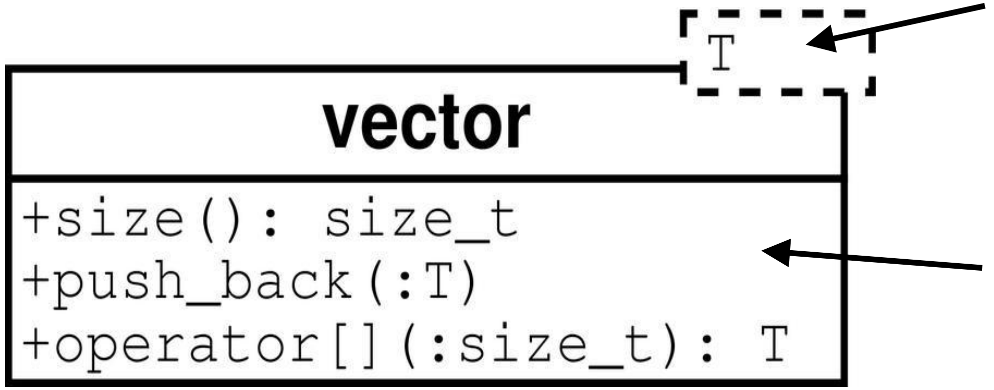
visibility name : type

2.2 Visibility

+ public	- private	# protected
Anyone can access	No-one can access	Subclasses can access
Interface operations	Data members	Operations where sub-classes collaborate
Not data members	Helper functions	Not data members (creates dependency off subclass on im-plementation of parent)
	"Friends" are allowd in though	

2.2 Template Classes

Generic classes depending on parametrised types



Type parameter(s)

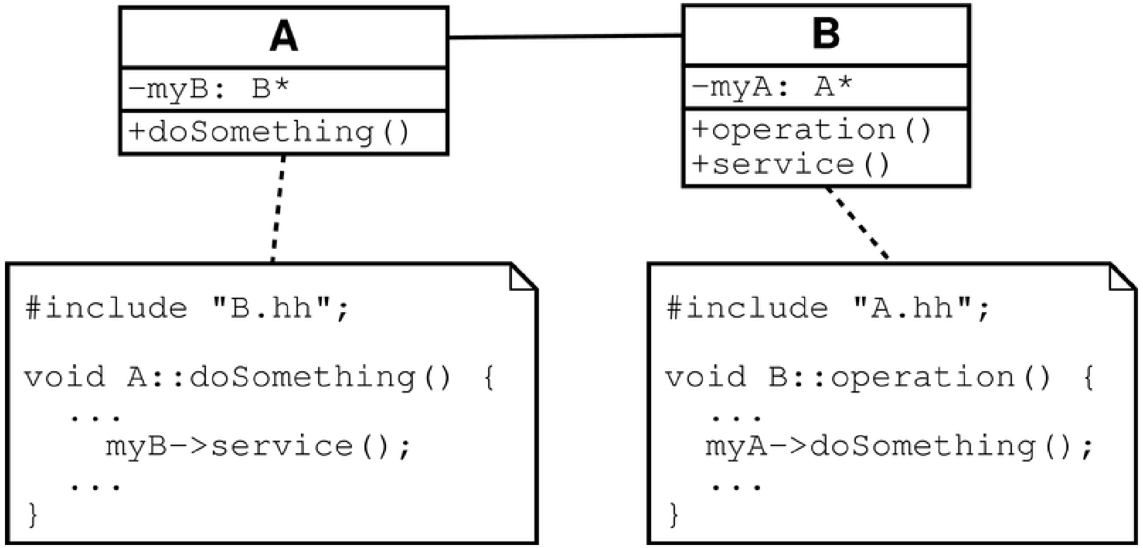
Operations compartment as usual, but may have type parameter instead of concrete type

2.3 Relations

- Association
- Aggregation
- Composition
- Parametric and Friendship
- Inheritance

2.3 Binary Association

Binary association: both classes know each other



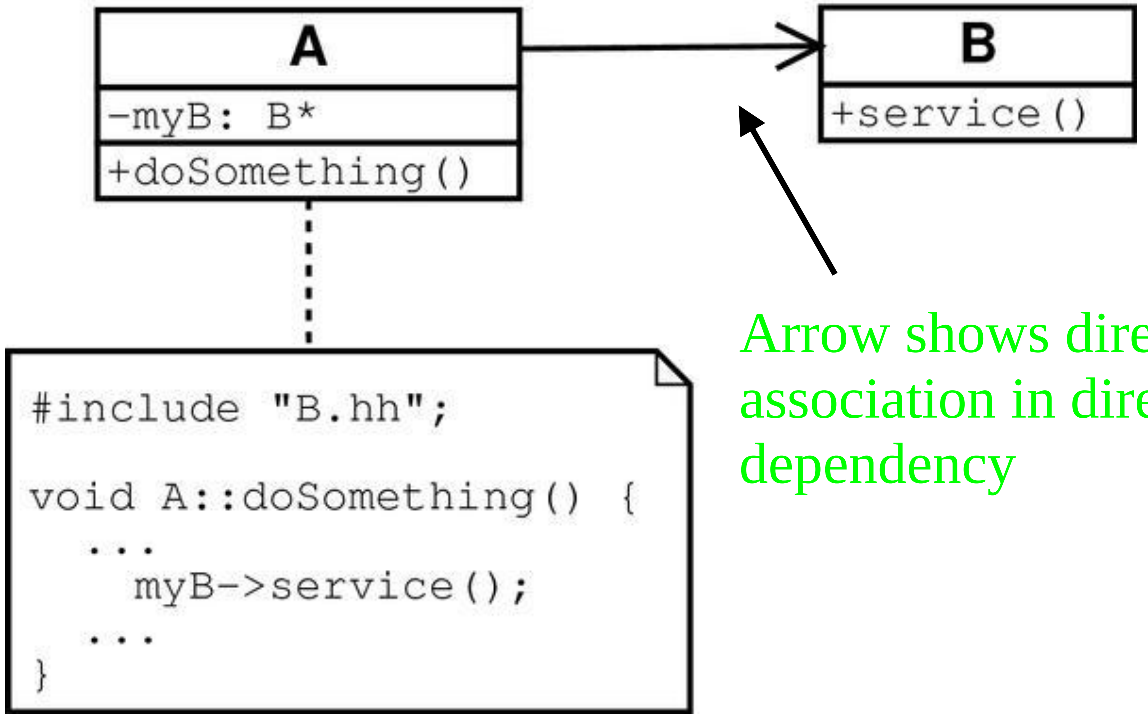
Usually "knows about" means a pointer or reference

Other methods possible: method argument, tables, database, ...

Implies dependency cycle

2.3 Unary Association

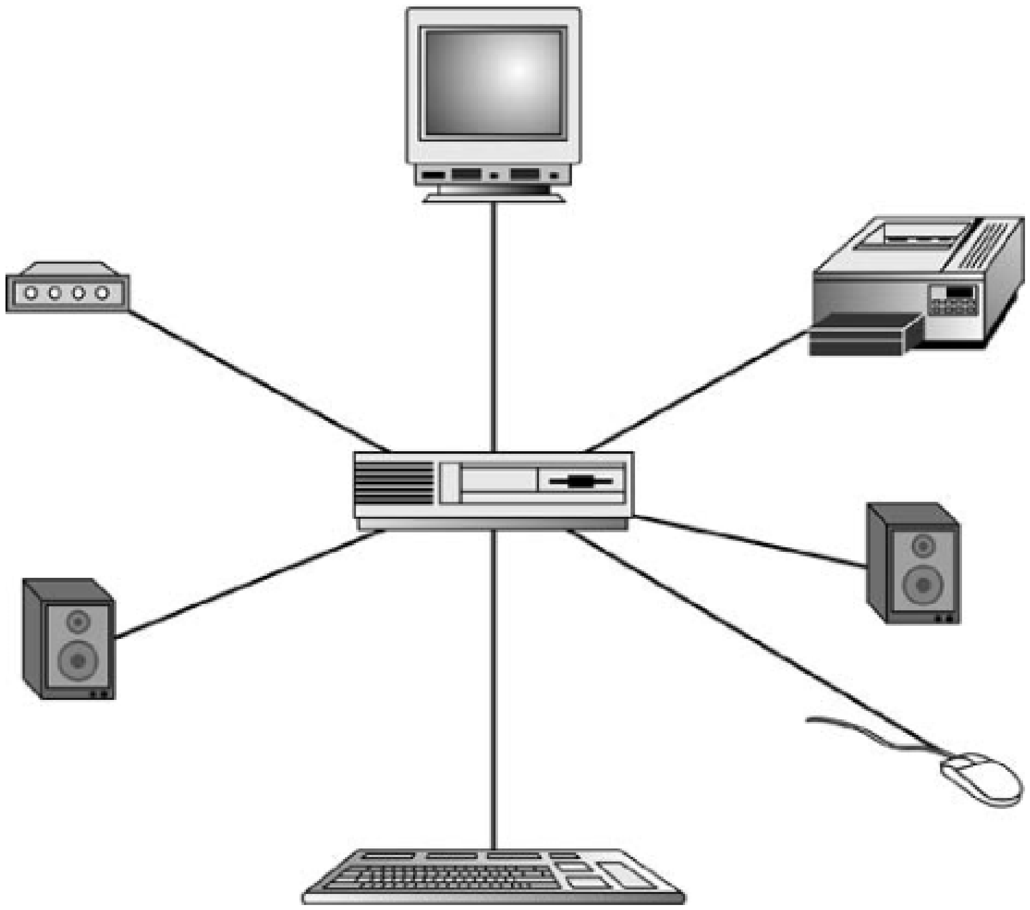
A knows about B, but B knows nothing about A



Arrow shows direction of association in direction of dependency

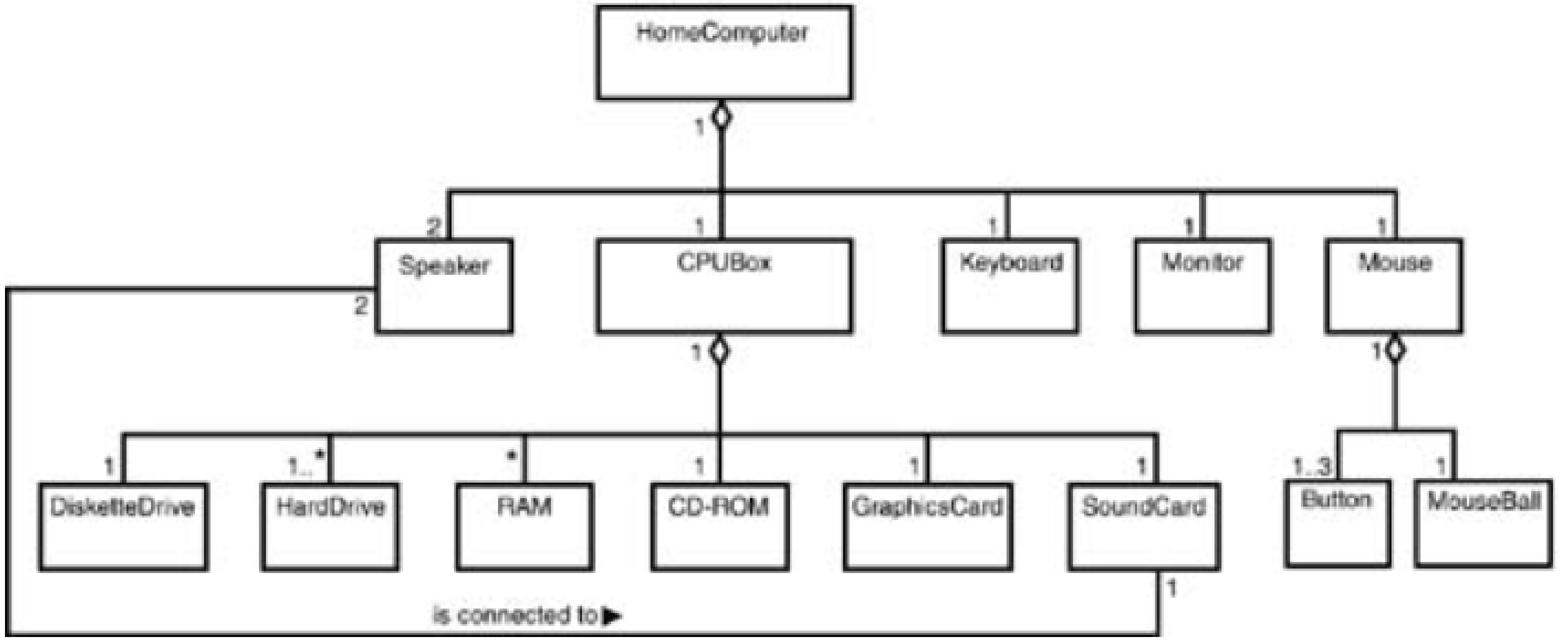
2.3 Aggregation

Figure 2.11. A typical computer system is an example of an aggregation—an object that's made up of a combination of a number of different types of objects.



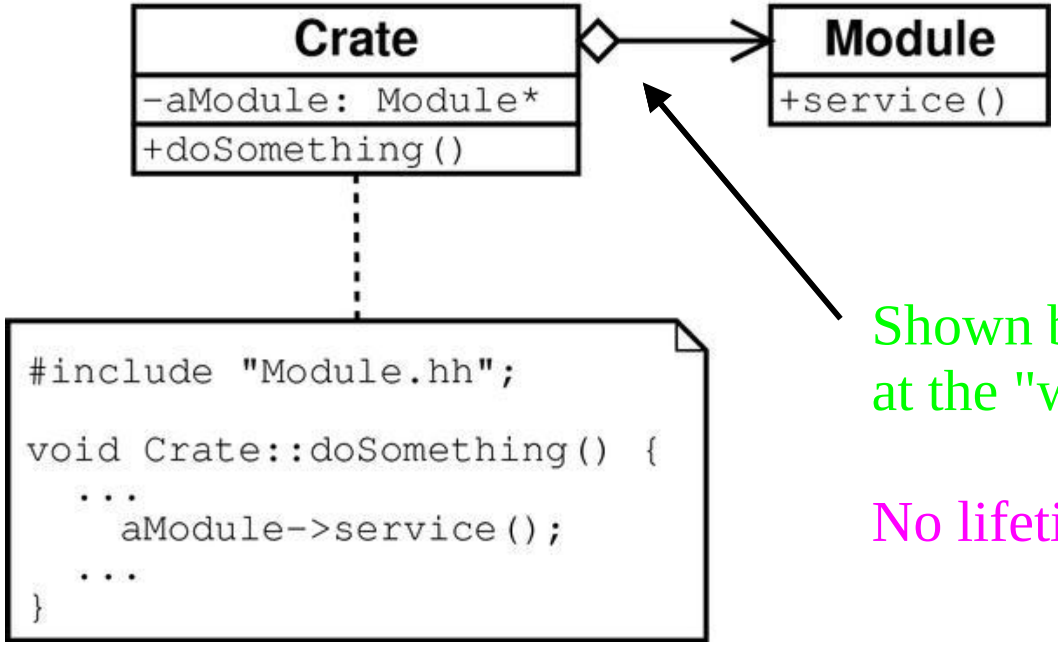
2.3 Aggregation

Figure 5.1. An aggregation (part-whole) association is represented by a line between the component and the whole with an open diamond adjoining the whole.



2.3 Aggregation

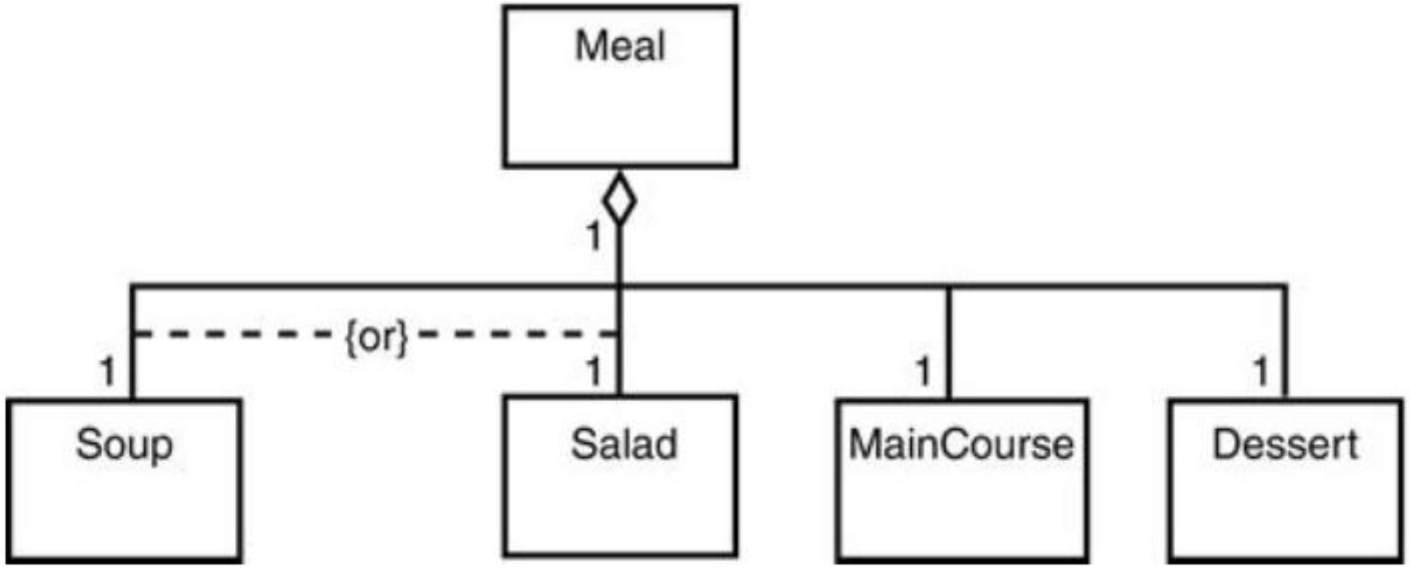
Aggregation = Association with "whole-part" relationship



Shown by hollow diamond at the "whole" side

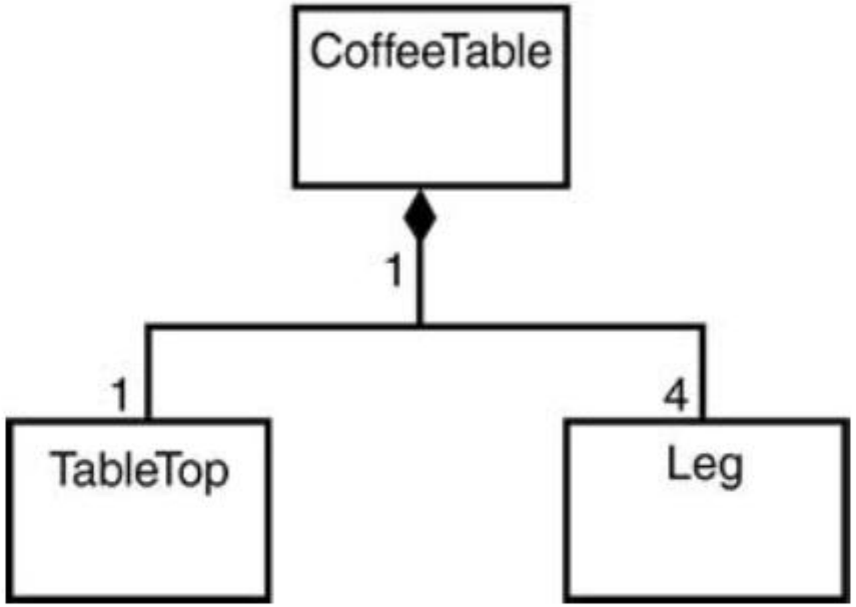
No lifetime control implied

2.3 Aggregation



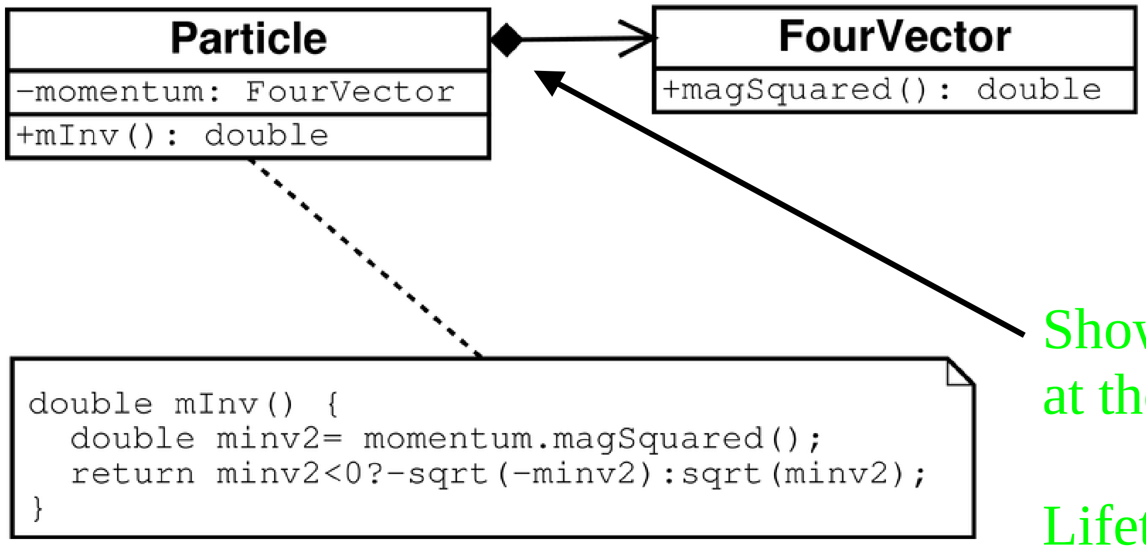
2.3 Composition

Figure 5.3. In a composite, each component belongs to exactly one whole. A closed diamond represents this relationship.



2.3 Composition

Composition = Aggregation with lifetime control



Shown by filled diamond at the "owner" side

Lifetime control implied

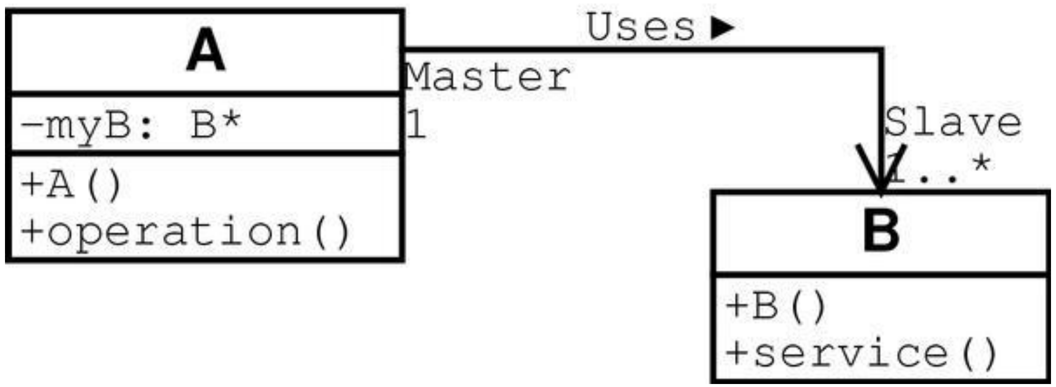
Lifetime control can be tranferred

Lifetime control: construction and destruction controlled by "owner"

→ call constructors and destructors (or have somebody else do it)

2.3 Association Details

Name gives details of association
Name can be viewed as verb of a sentence

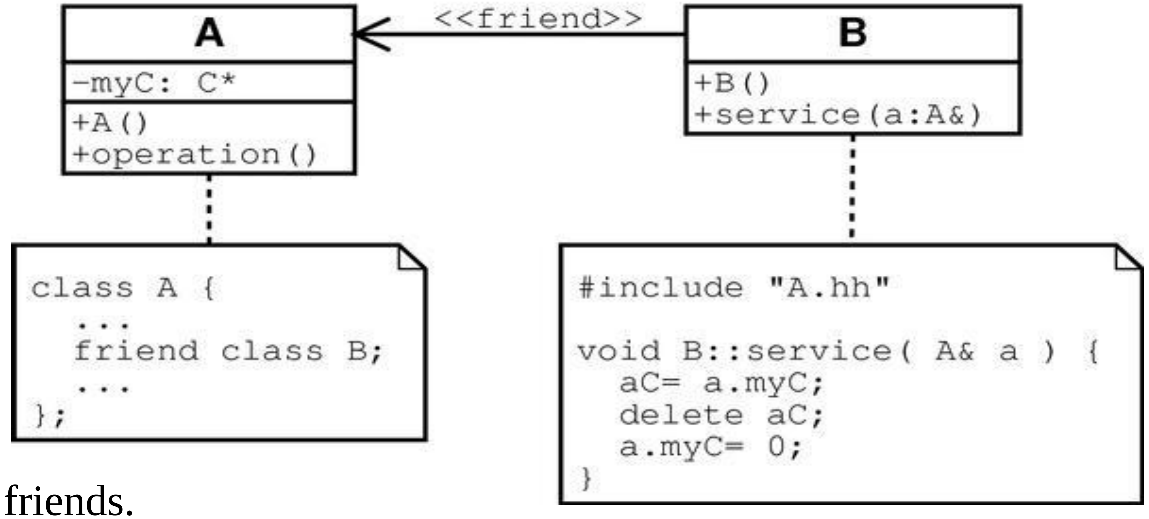


Notes at association ends explain "roles" of classes (objects)

Multiplicities show number of objects which participate in the association

2.3 Friendship

Friends are granted access to private data members and member functions
Friendship is given to other classes, never taken

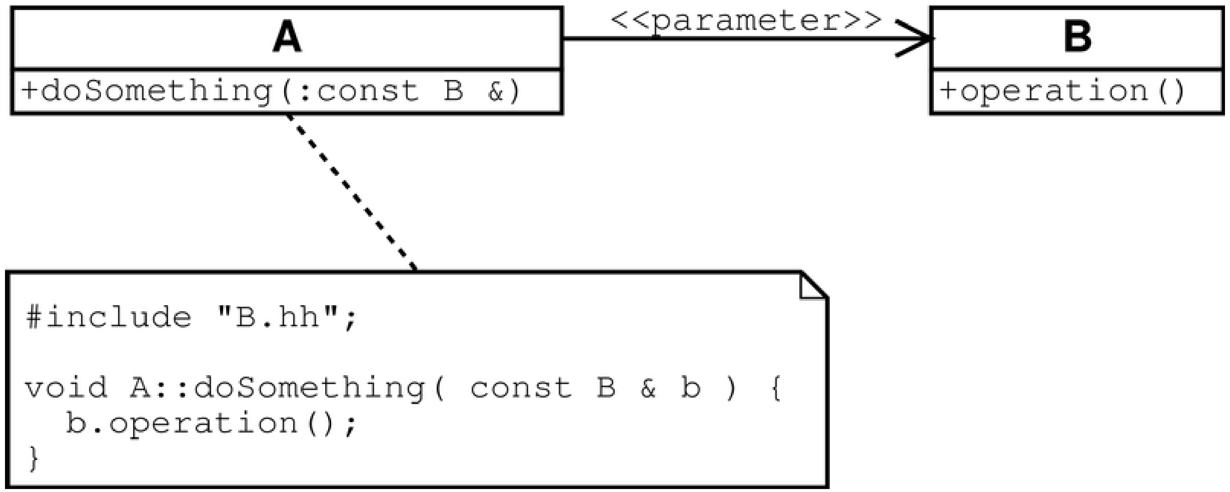


Bob Martin:
More like lovers than friends.
You can have many friends,
you should not have many lovers

Friendship breaks data hiding, use carefully

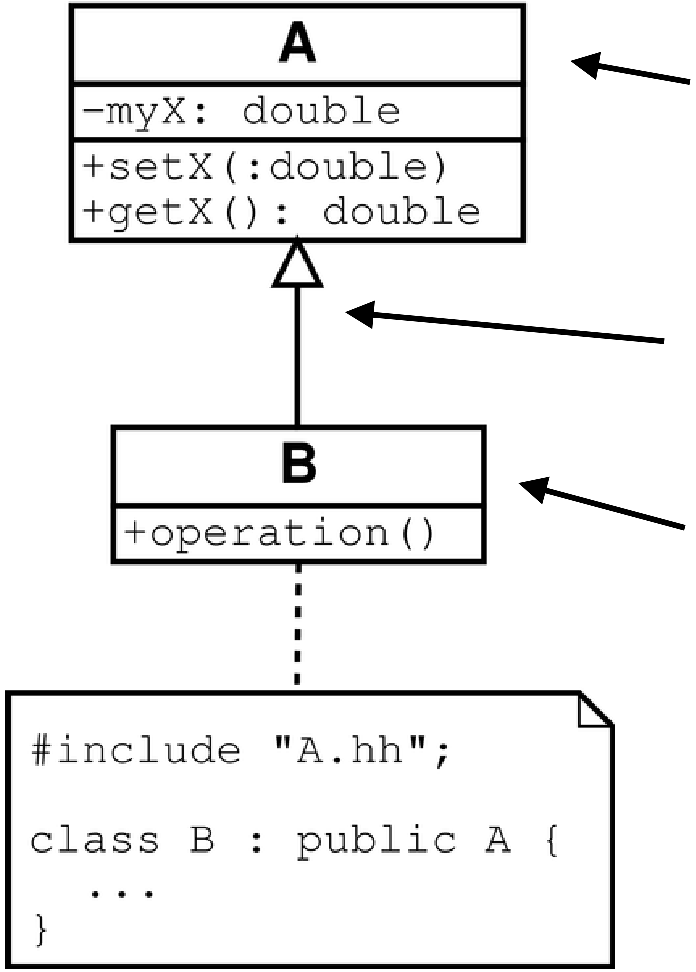
2.3 Parametric Association

Association mediated by a parameter (function call argument)



A depends upon B, because it uses B
No data member of type B in A

2.3 Inheritance



Base class or super class

Arrow shows direction
of dependency

Derived class or subclass

→ B inherits A's interface, behaviour and data members

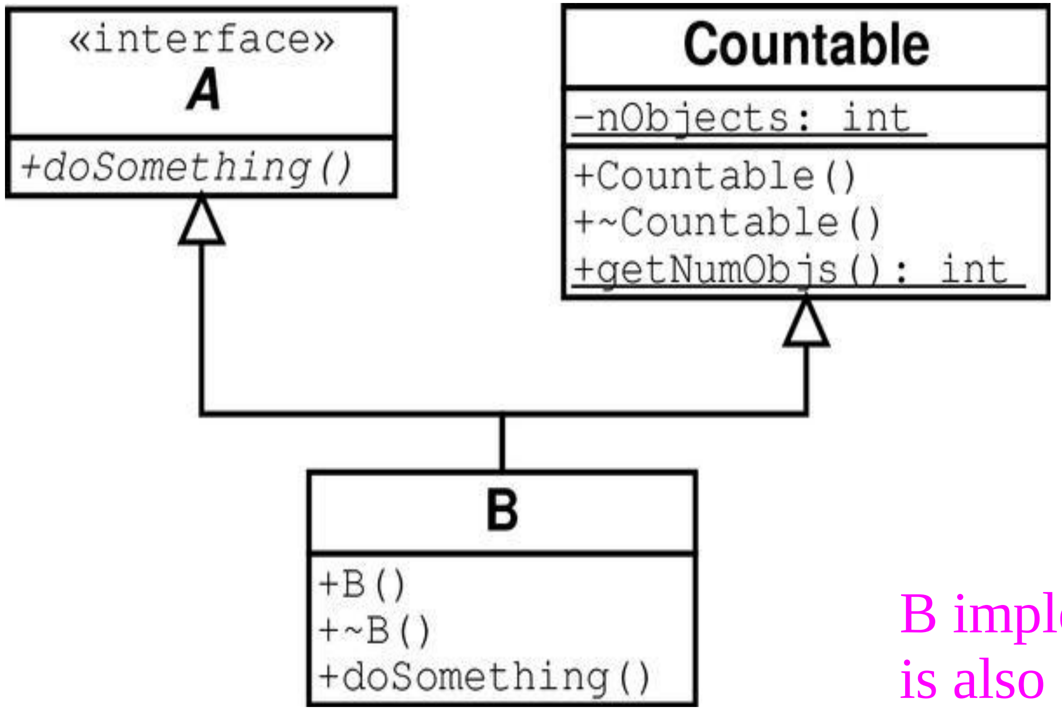
→ B can extend A, i.e. add new data members or member functions

→ B depends on A, A knows nothing about B

2.3 Associations Summary

- Can express different kinds of associations between classes/objects with UML
  - Association, aggregation, composition, inheritance
  - Friendship, parametric association
- Can go from simple sketches to more detailed design by adding *adornments*
  - *Name, roles, multiplicities*
  - *lifetime control*

2.3 Multiple Inheritance



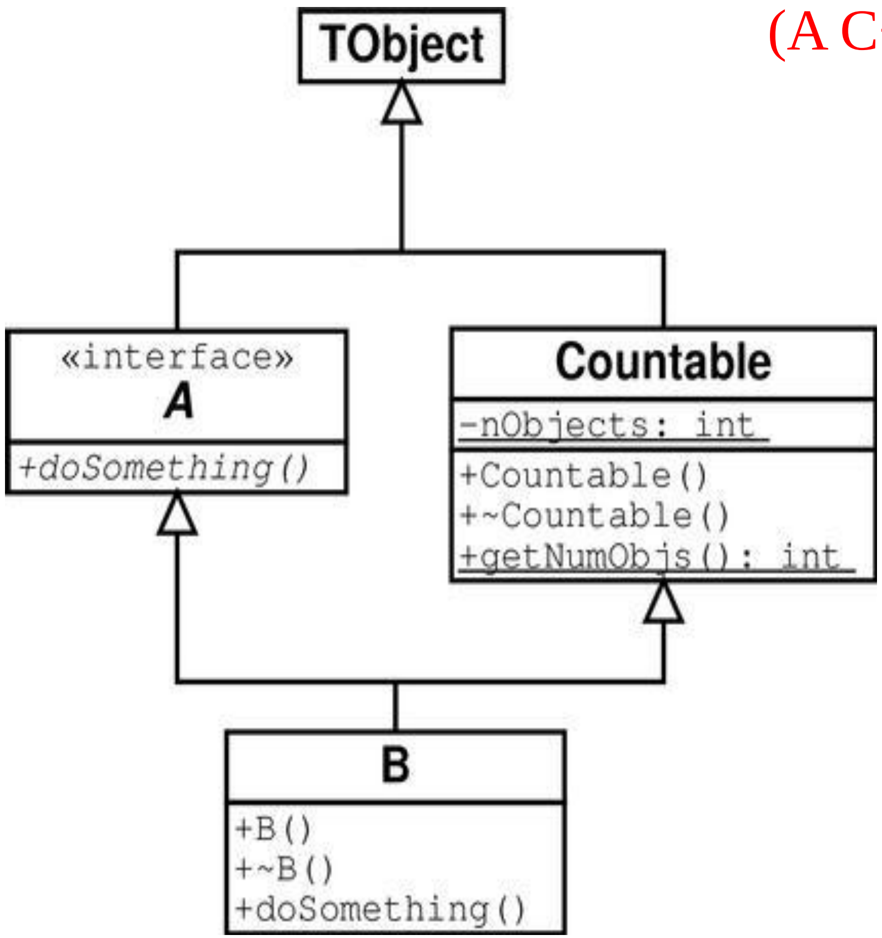
The derived class inherits interface, behaviour and data members of all its base classes

Extension and overriding works as before

B implements the interface A and is also a "countable" class

Countable also called a "Mixin class"

2.3 Deadly Diamond of Death



(A C++ feature)

Now the @*#! hits the %&$?

Data members of TObject are inherited twice in B, which ones are valid?

Fortunately, there is a solution to this problem:

→ virtual inheritance in C++: only one copy of a multiply inherited structure will be created

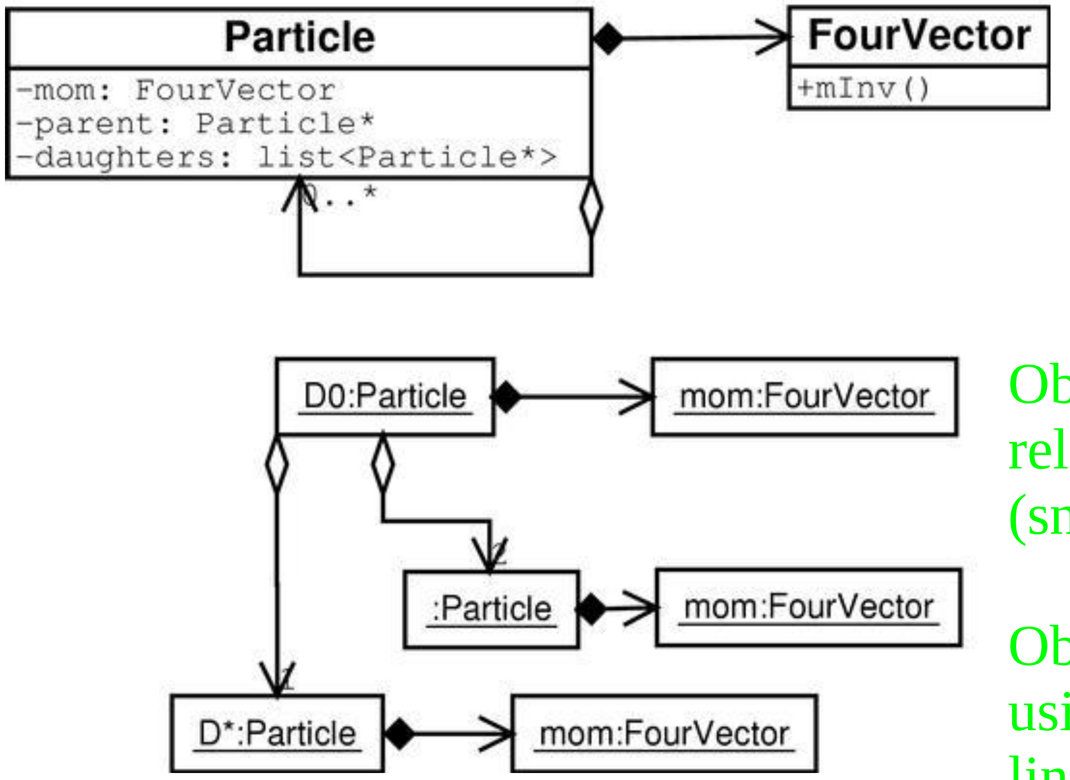
2.4 Static and Dynamic Design

- Static design describes code structure and object relations
  - Class relations
  - Objects at a given time
- Dynamic design shows communication between objects
  - Similarity to class relations
  - can follow sequences of events

2.4 Class Diagram

- Show static relations between classes
  - we have seen them already
  - interfaces, data members
  - associations
- Subdivide into diagrams for specific purpose
  - showing all classes usually too much
  - ok to show only relevant class members
  - set of all diagrams should describe system

2.4 Object Diagram



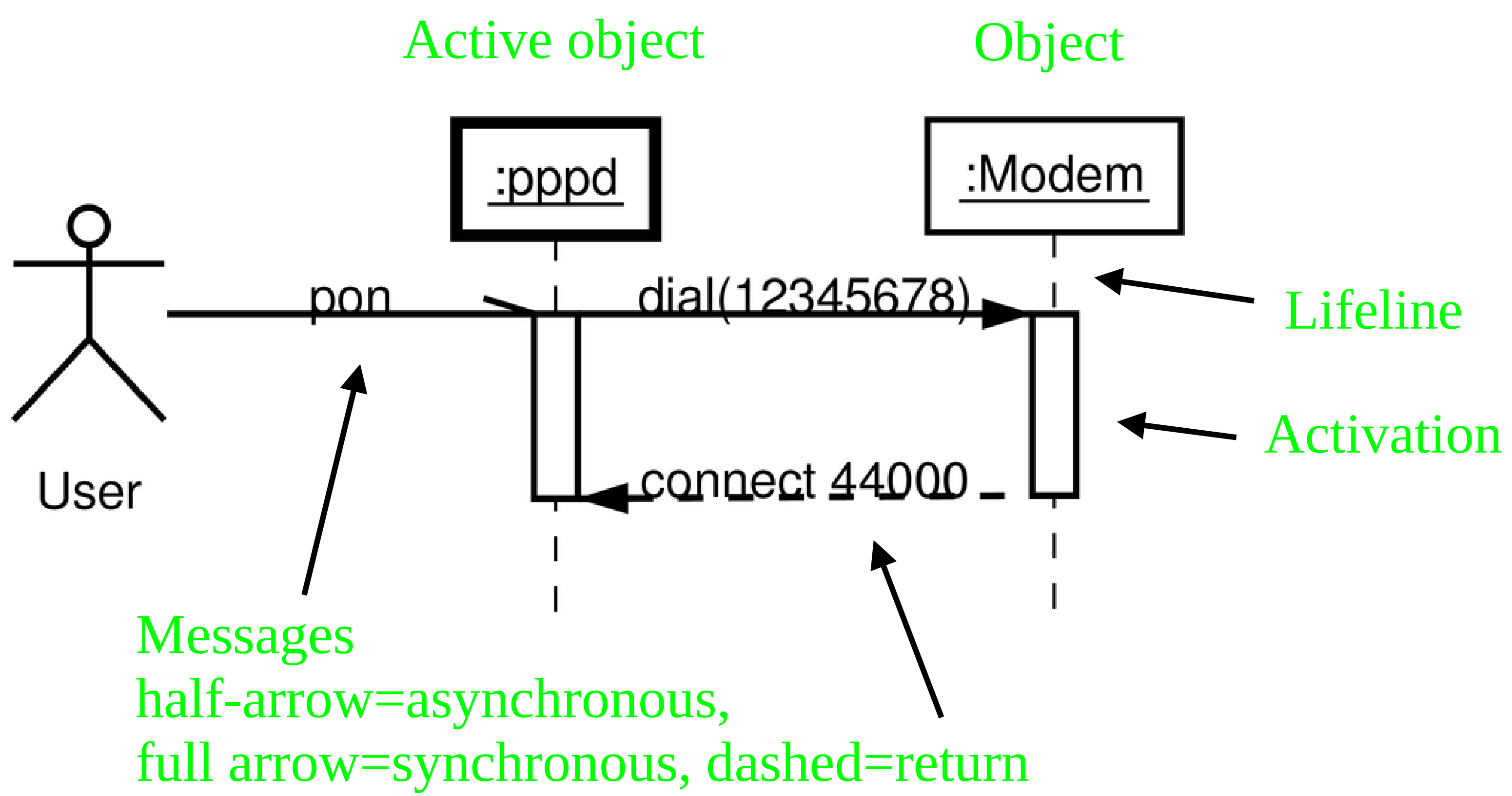
Class diagram never changes

Object diagram shows relations at instant in time (snapshot)

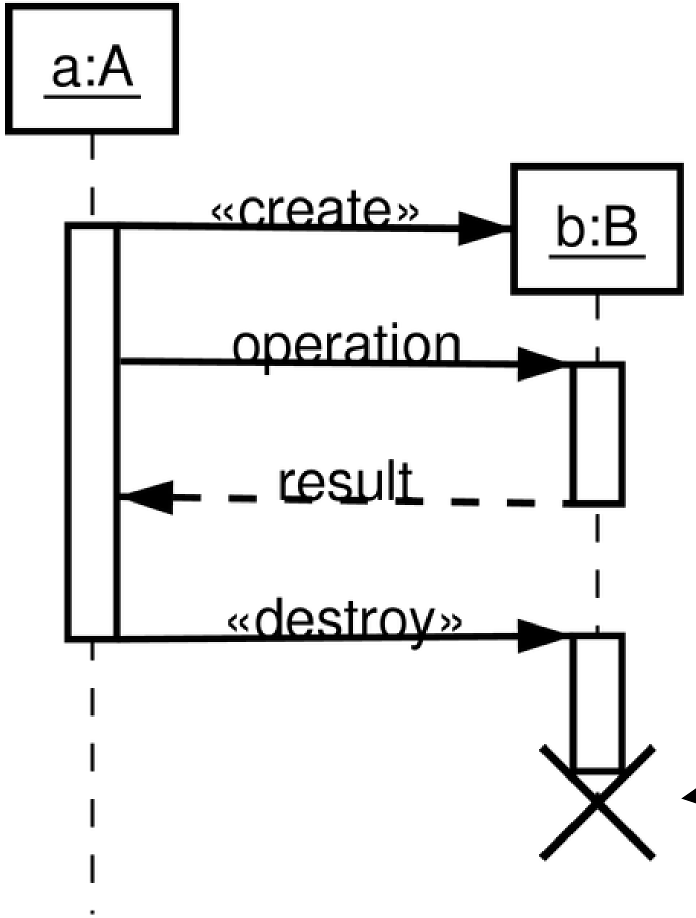
Object relations are drawn using the class association lines

2.4 Sequence Diagram

Show sequence of events for a particular use case



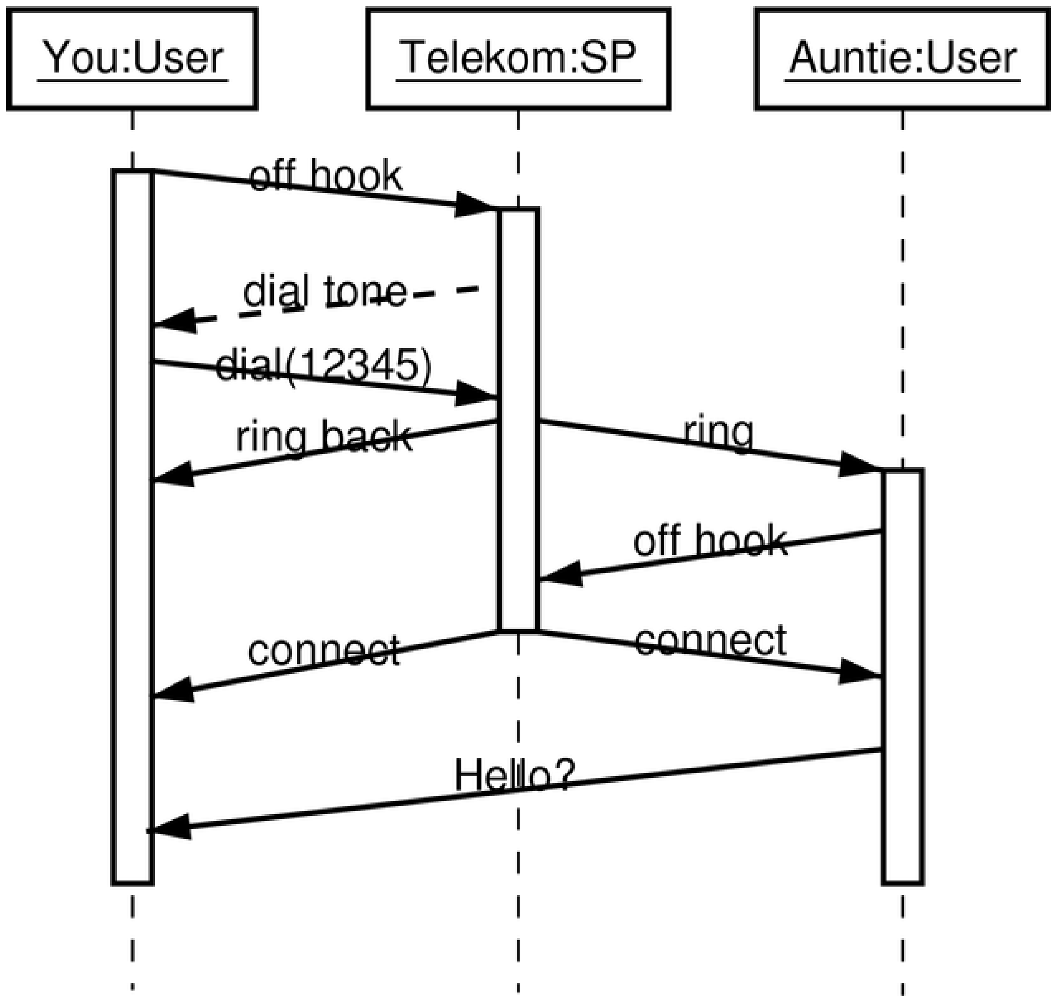
2.4 Sequence Diagram



Can show creation and destruction of objects

Destruction mark

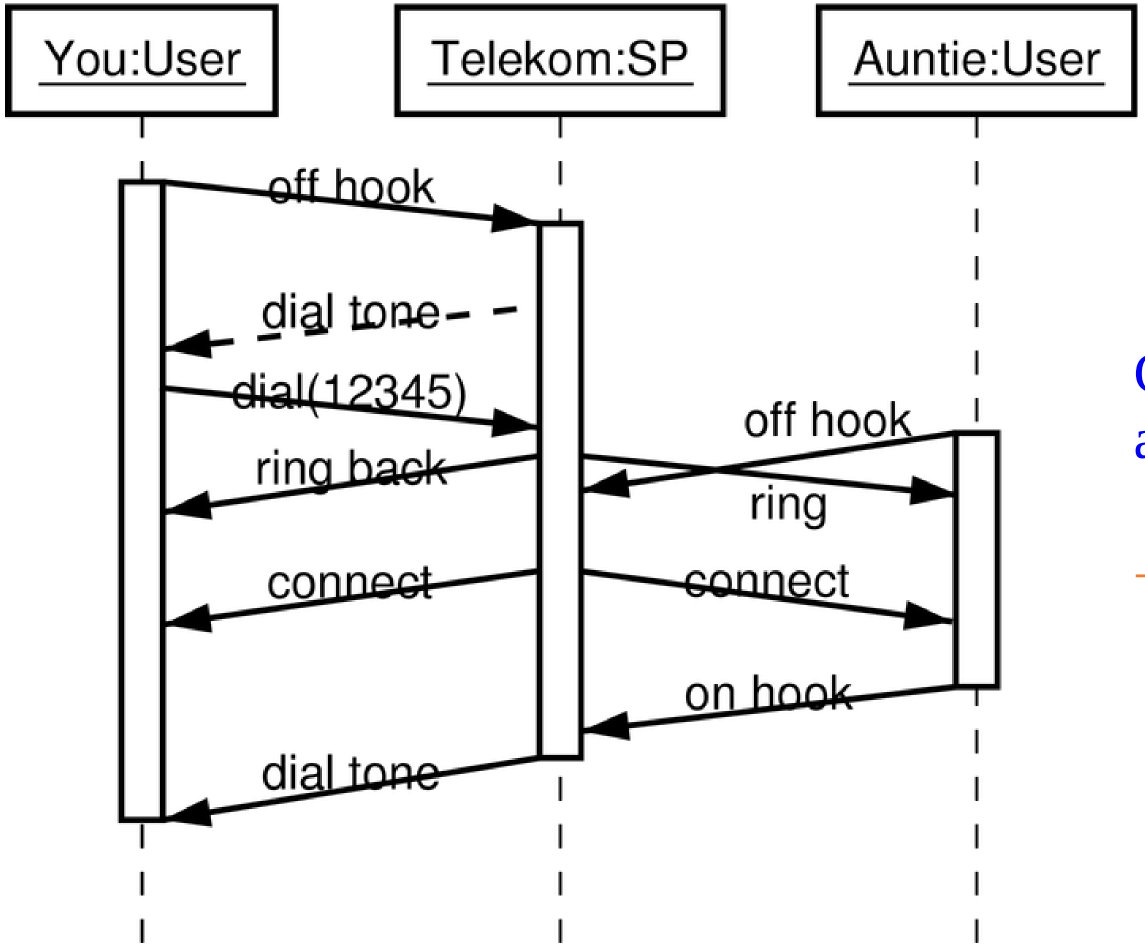
2.4 Sequence Diagram



Slanted messages take some time

Can model real-time systems

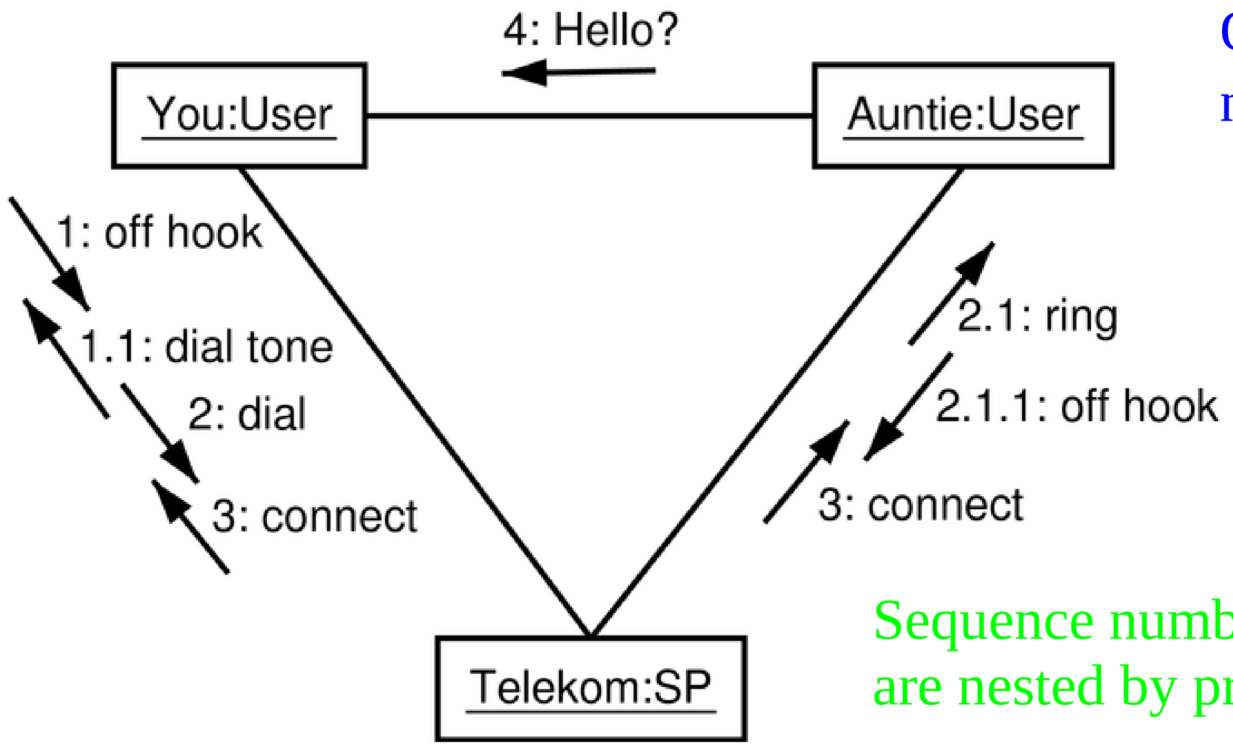
2.4 Sequence Diagram



Crossing message lines are a bad sign

→ race conditions

2.4 Collaboration Diagram



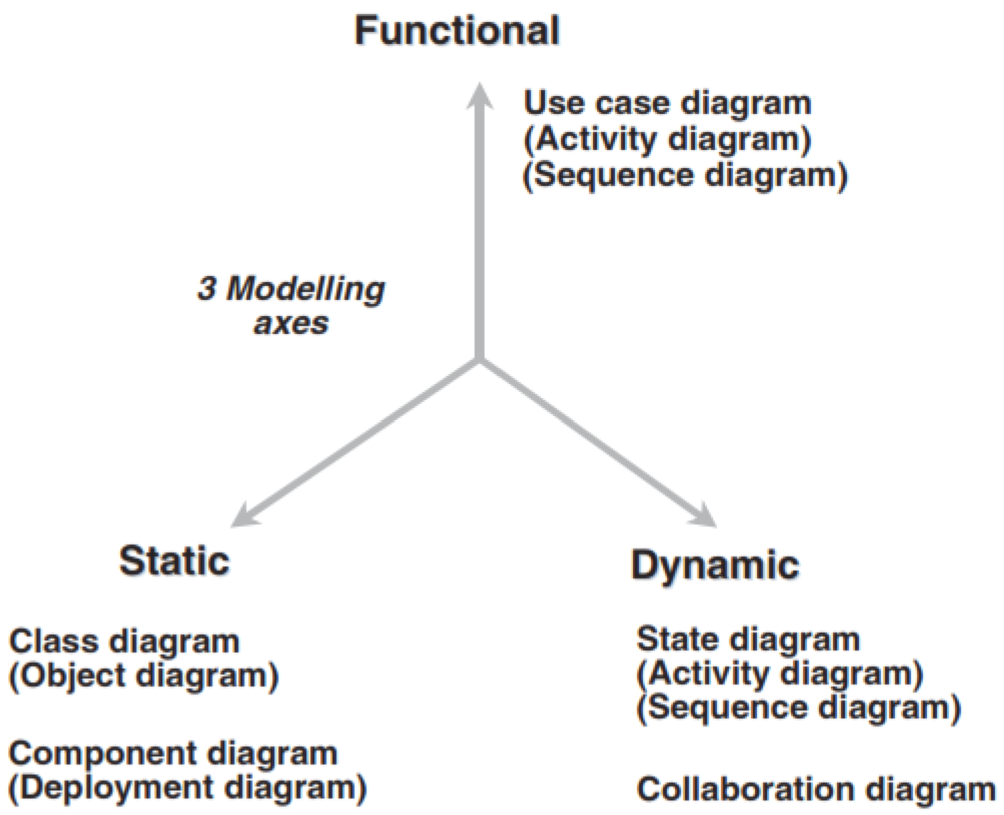
Object diagram with numbered messages

Sequence numbers of messages are nested by procedure call

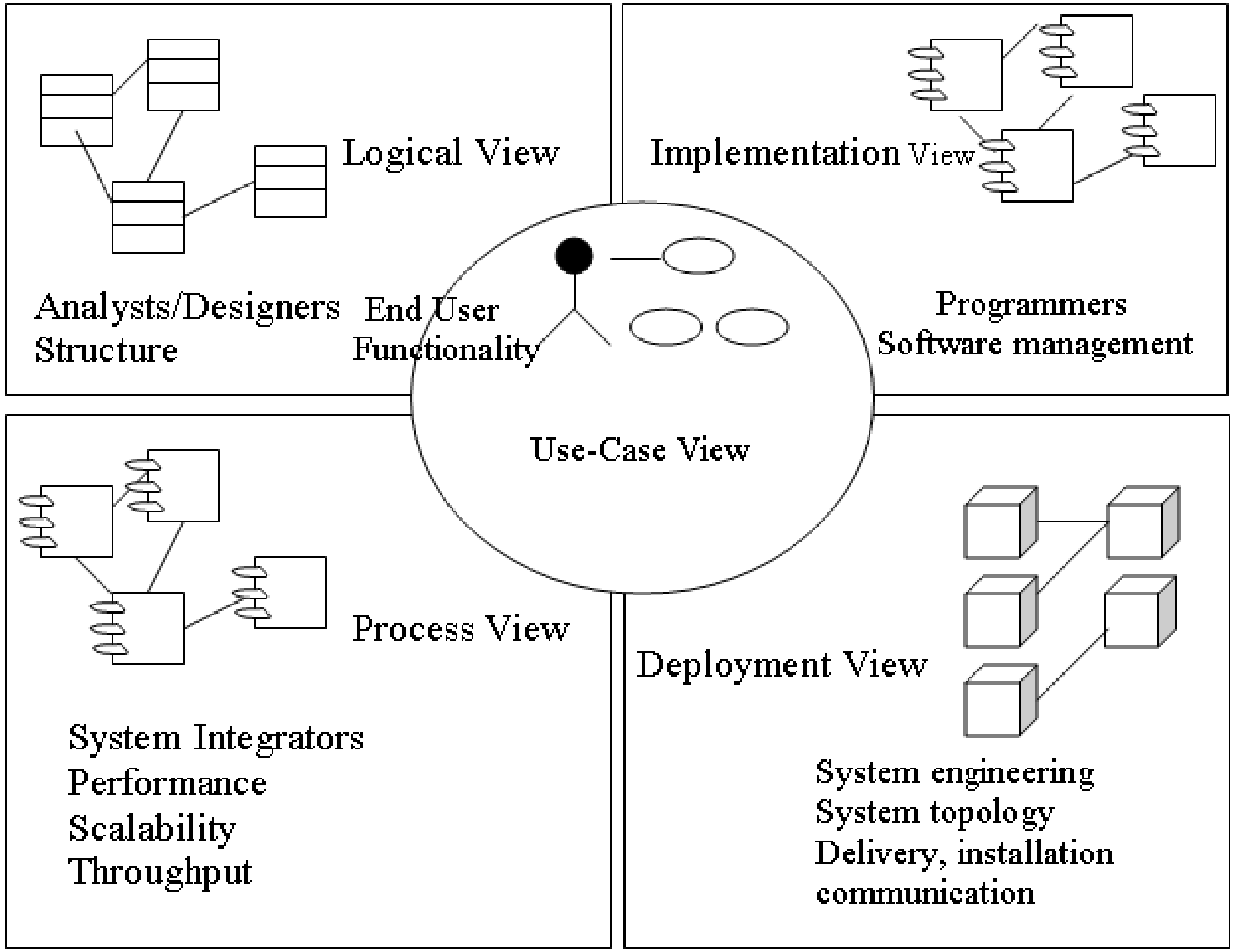
2.4 Static and Dynamic Design Summary

- Class diagrams → object diagrams
  - classes → objects; associations → links
- Dynamic models show how system works
  - Sequence and collaboration diagram
- There are tools for this process
  - UML syntax and consistency checks
- Rough sketches by hand or with simple tools
  - aid in design discussions

Modelling Aspects



"4+1" View



Logical View
Analysts/Designers
Structure
Implementation View
Programmers
Software management
End User
Functionality
Use-Case View
Process View
System Integrators
Performance
Scalability
Throughput
Deployment View
System engineering
System topology
Delivery, installation
communication

Case Study – ATM (Automatic Teller Machine)

Problem Statement-The ATM offers the following services:

1) Distribution of money to every holder of a smartcard via a card reader and a cash dispenser.
2) Consultation of account balance, cash and cheque deposit facilities for bank customers who hold a smartcard from their bank.
3) All transactions are made secure.
4) It is sometimes necessary to refill the dispenser, etc.

Steps we should take:

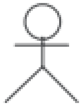
- Identify the actors,
- Identify the use cases,
- Construct a use case diagram,
- Write a textual description of the use cases,
- Complete the descriptions with dynamic diagrams,
- Organize and structure the use cases.

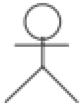
Case Study – ATM (Automatic Teller Machine)

Step #1: Identifying the actors of the ATM

Problem Statement-The ATM offers the following services:

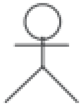
1) Distribution of money to every holder of a smartcard via a card reader and a cash dispenser.
2) Consultation of account balance, cash and cheque deposit facilities for bank customers who hold a smartcard from their bank.
3) All transactions are made secure.
4) It is sometimes necessary to refill the dispenser, etc.

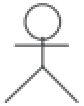




Maintenance operator

- VISA AS (Issuer) for withdrawal transactions carried out using a Visa smartcard
- Bank IS (Acquirer) to authorize all transactions carried out by a customer using his or her bank smartcard, but also to access the account balance.

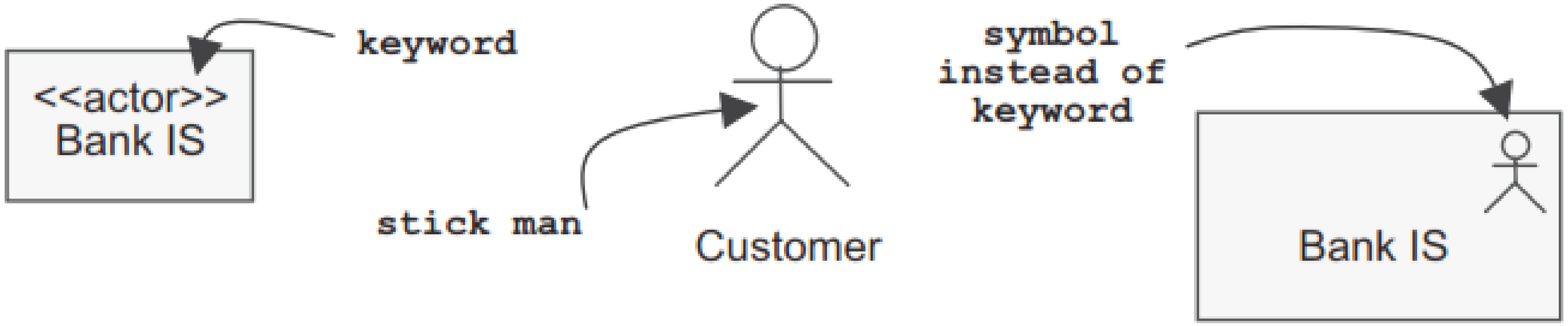




An interview with the domain expert

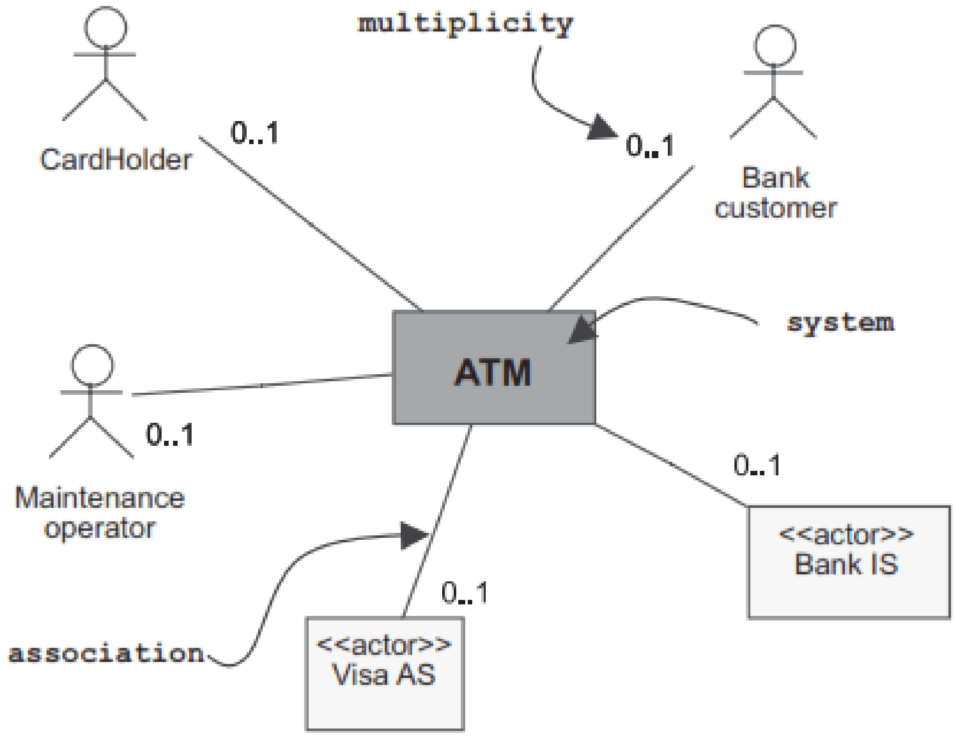
Case Study – ATM (Automatic Teller Machine)

Possible graphical representations of an actor



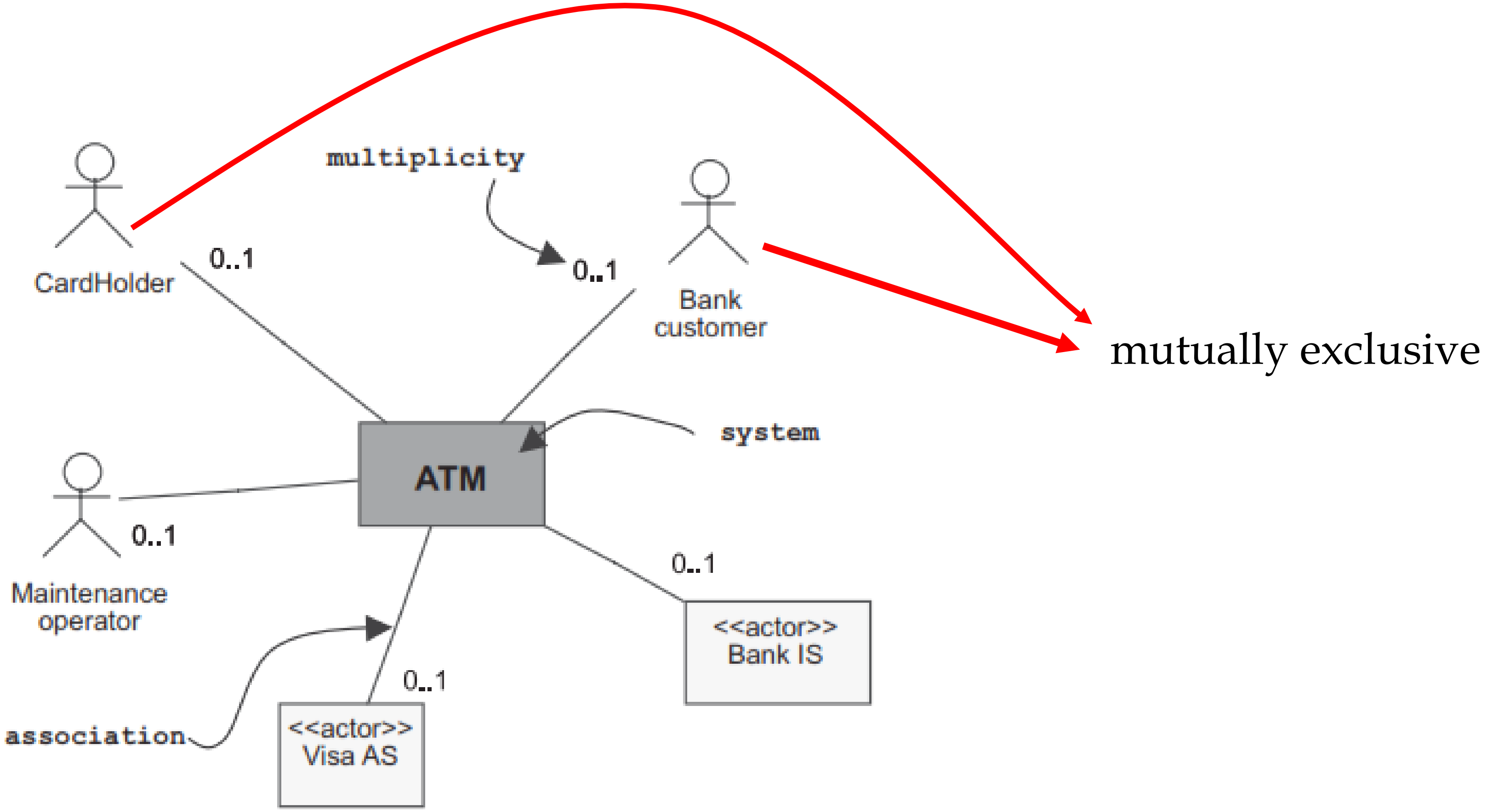
Case Study – ATM (Automatic Teller Machine)

Static Context Diagram



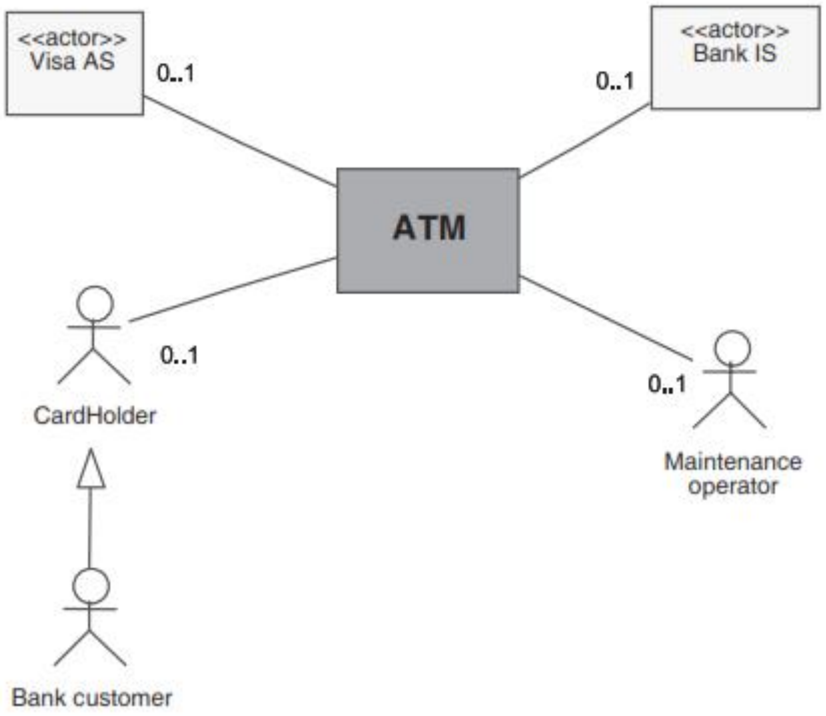
Case Study – ATM (Automatic Teller Machine)

Static Context Diagram



Case Study – ATM (Automatic Teller Machine)

Static Context Diagram



*Bank customer as a **specialization** of CardHolder*

Case Study – ATM (Automatic Teller Machine)

Step #2: Identifying use cases

- A use case represents the specification of a **sequence of actions**, including variants, that a system can perform, *interacting with actors* of the system.
- A use case models a **service** offered by the system. It expresses the actor/system interactions and yields an observable result of value to an actor.
- For each actor identified previously, it is advisable to search for the different **business goals**, according to which is using the system.

Case Study – ATM (Automatic Teller Machine)

Step #2: Identifying use cases

Prepare a preliminary list of use cases of the ATM:

primary actors

CardHolder:
- Withdraw money.

Bank customer:
- Withdraw money (something not to forget!).
- Consult the balance of one or more accounts.
- Deposit cash.
- Deposit cheques.

Maintenance operator:
- Refill dispenser.
- Retrieve cards that have been swallowed.
- Retrieve cheques that have been deposited.

Secondary actors (non-human)

Supporting actors

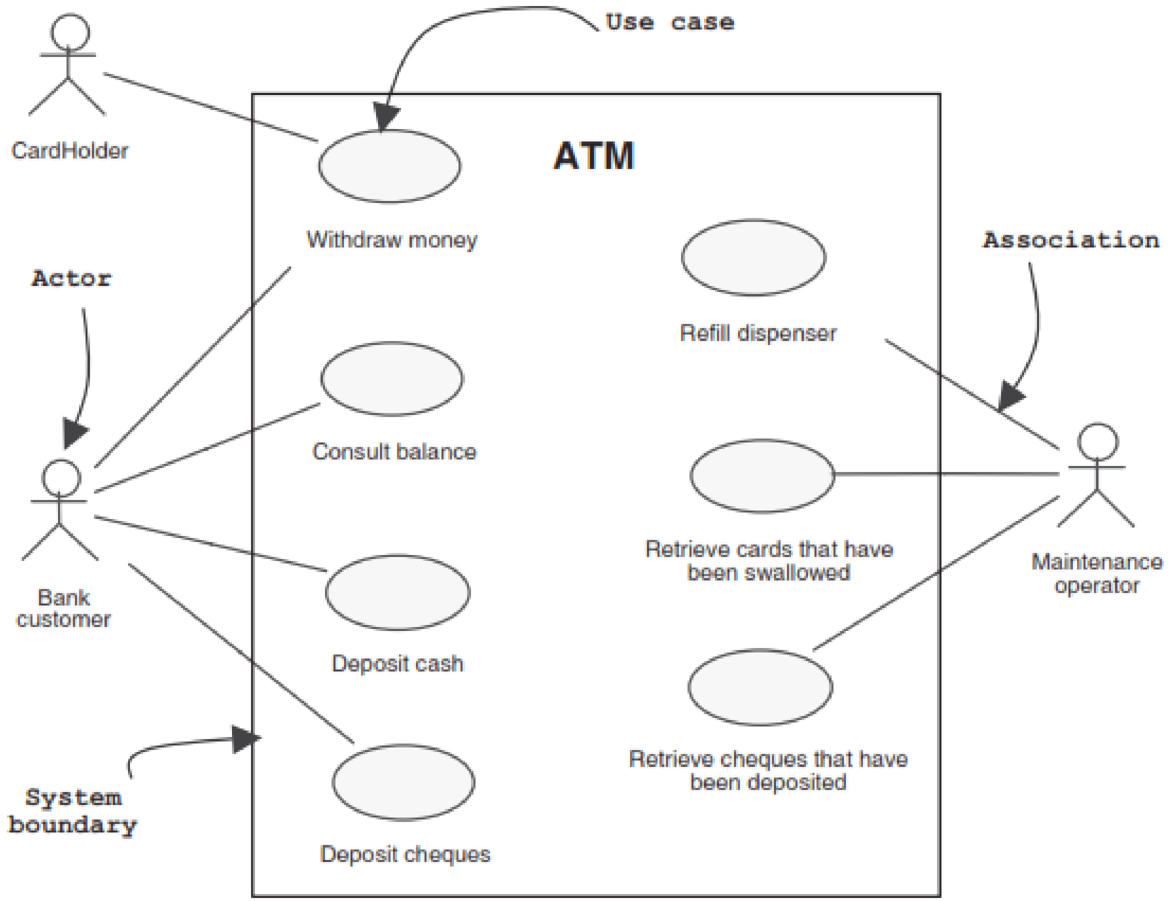
External actor

Visa authorization system (AS):
- None.

Bank information system (IS):
- None.

Case Study – ATM (Automatic Teller Machine)

Step #3: Creating use case diagrams

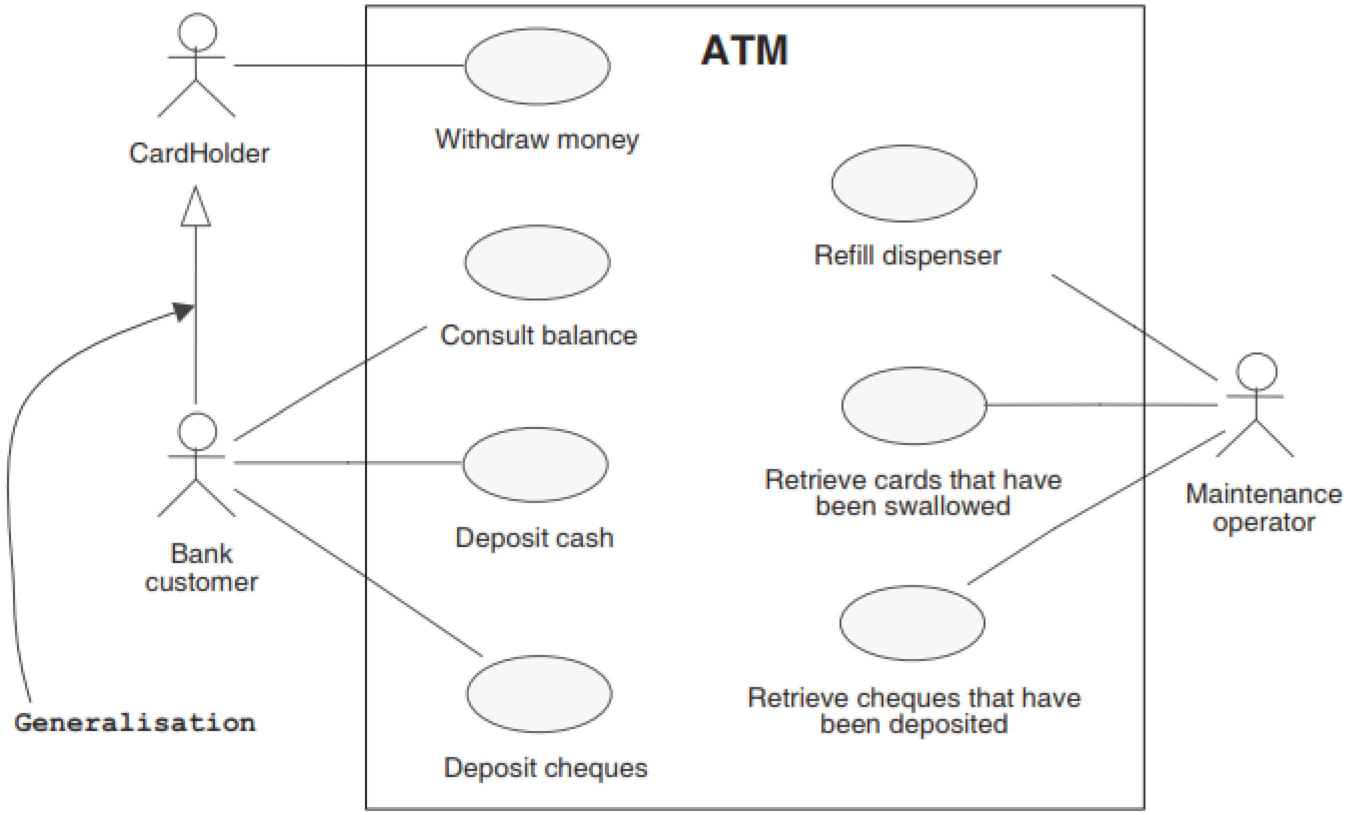


Preliminary use case diagram of the ATM

Case Study – ATM (Automatic Teller Machine)

Step #3: Creating use case diagrams

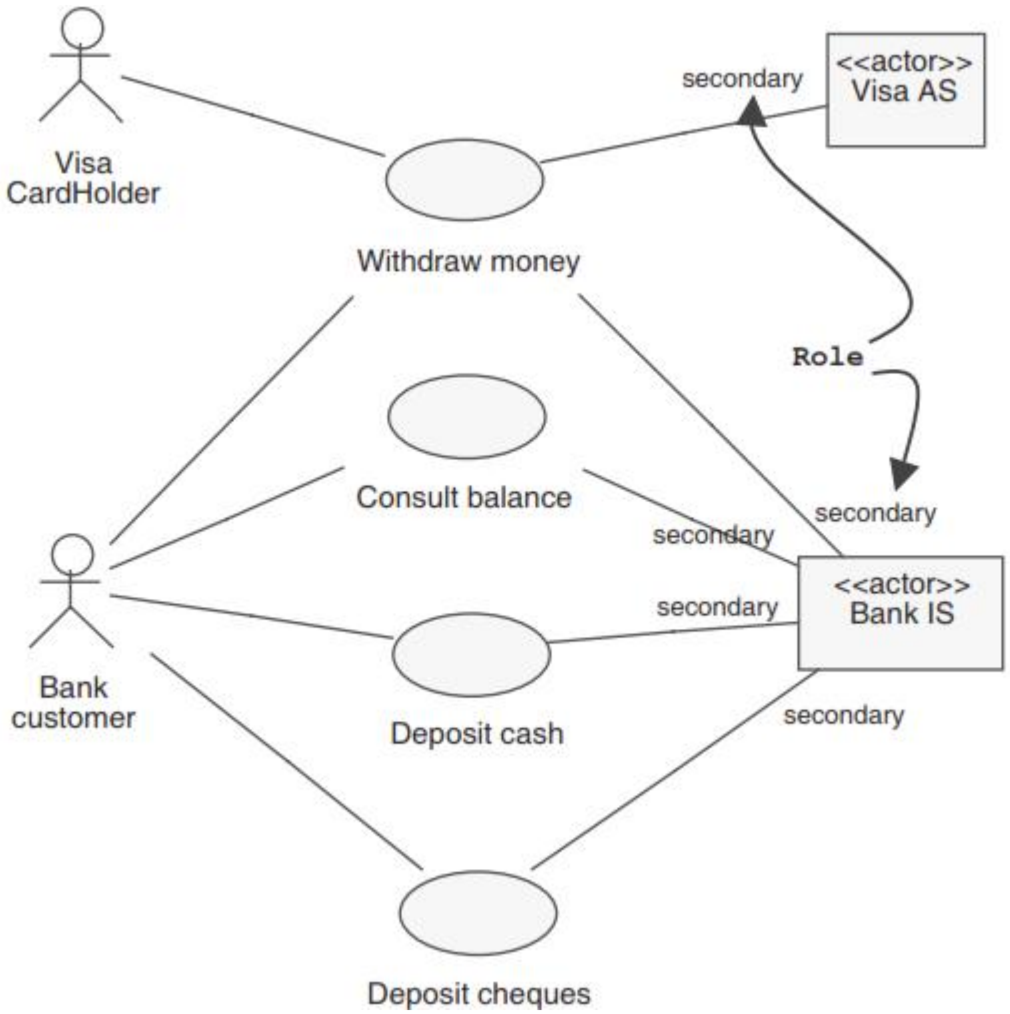
Generalization
Specialization
Inheritance



A more sophisticated version of the preliminary use case diagram

Case Study – ATM (Automatic Teller Machine)

Step #3: Creating use case diagrams



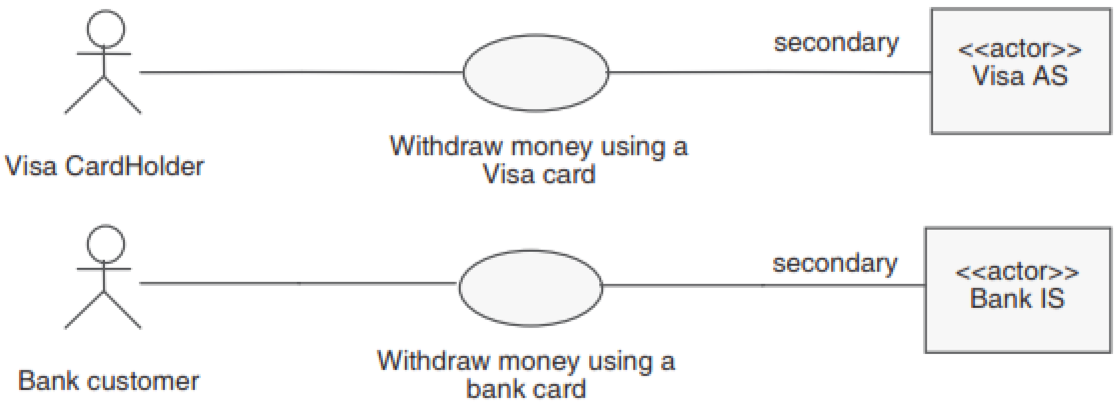
primary actors *to the* ***left*** *of the use cases and the*

secondary actors *to the* ***right****.*

Simple version of the completed use case diagram

Case Study – ATM (Automatic Teller Machine)

Step #3: Creating use case diagrams



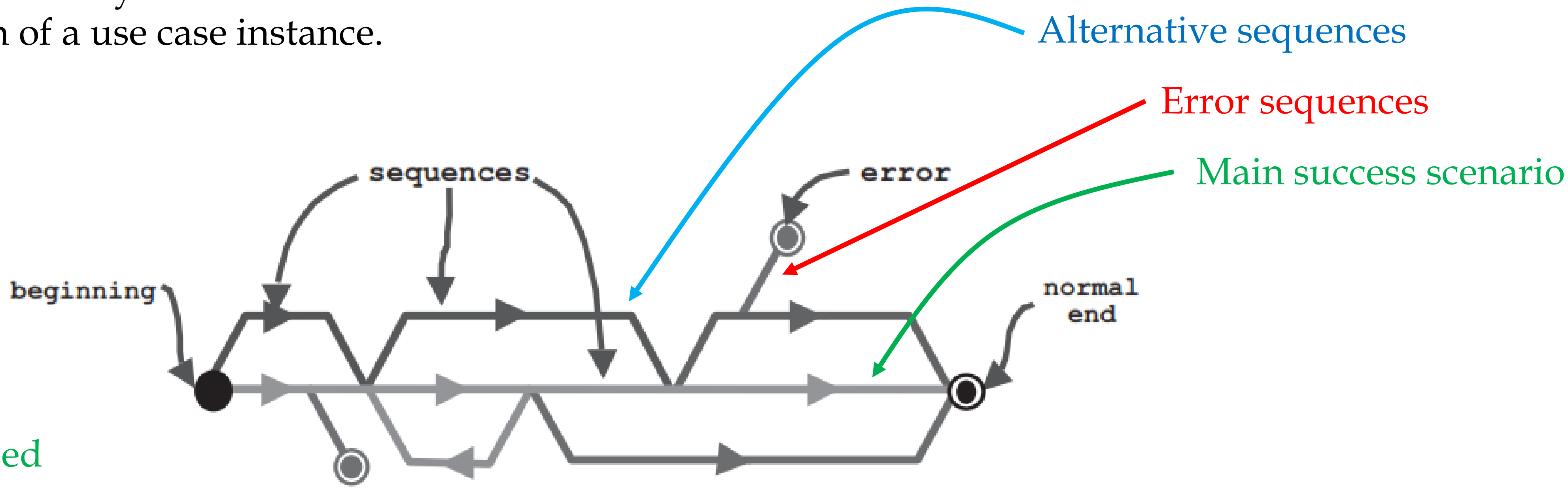
If the 2 use cases cannot occur at the same time…

Representation of the scenarios of a use case

Case Study – ATM (Automatic Teller Machine)

Step #4: Textual description of use cases

We call each unit of description of action sequences a ***sequence***. *A* ***scenario*** represents a particular succession of sequences, which is run from beginning to end of the use case. A scenario may be used to illustrate an interaction or the execution of a use case instance.



* Not standardized in UML

Representation of the scenarios of a use case

Case Study – ATM (Automatic Teller Machine)

Step #4: Textual description of use cases

separating the actions of the **actors** and those of the **system** into two columns

1. The Visa CardHolder inserts his or her card in the ATM's card reader.	2. The ATM verifies that the card that has been inserted is indeed a Visa card. 3. The ATM asks the Visa CardHolder to enter his or her pin number.
4. The Visa CardHolder enters his or her pin number.	5. The ATM compares the pin number with the one that is encoded on the chip of the card. 6. The ATM requests an authorisation from the VISA authorisation system.
7. The VISA authorisation system confirms its agreement and indicates the daily balance.	8. The ATM asks the Visa CardHolder to enter the desired withdrawal amount.
9. The Visa CardHolder enters the desired withdrawal amount.	10. The ATM checks the desired amount against the daily balance. 11. The ATM asks the Visa CardHolder if he or she would like a receipt.
12. The Visa CardHolder requests a receipt.	13. The ATM returns the card to the Visa CardHolder.
14. The Visa CardHolder takes his or her card.	15. The ATM issues the notes and a receipt.
16. The Visa CardHolder takes the notes and the receipt.	

Pre-conditions:

- The ATM cash box is well stocked.
- There is no card in the reader.

Post-conditions:

- The cashbox of the ATM contains fewer notes than it did at the start of the use case.

Case Study – ATM (Automatic Teller Machine)

Step #4: Textual description of use cases

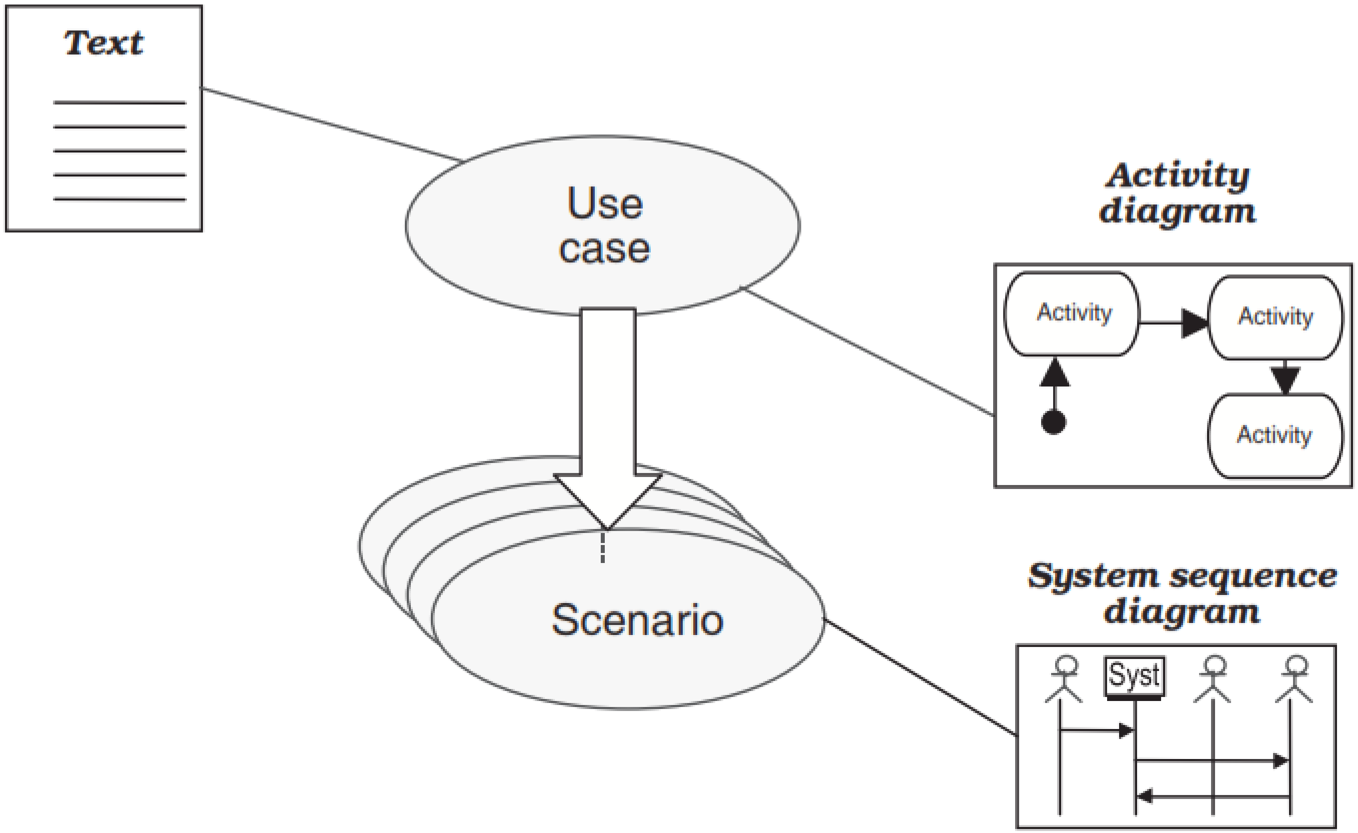
Non-functional constraints

Constraints	Specifications
Response time	The interface of the ATM must respond within a maximum time limit of 2 seconds. A nominal withdrawal transaction must take less than 2 minutes.
Concurrency	Non applicable (single user).
Availability	The ATM can be accessed 24/7.[14] A lack of paper for the printing of receipts must not prevent the card holder from being able to withdraw money.
Integrity	The interfaces of the ATM must be extremely sturdy to avoid vandalism.
Confidentiality	The procedure of comparing the pin number that has been entered on the keyboard of the ATM with that of the smartcard must have a maximum failure rate of 10^{-6}.

❖ This non-functional requirement is here as an example, but should be removed in the end and put at the system level as it applies to all use cases.

Case Study – ATM (Automatic Teller Machine)

Step #5: Graphical description of use cases



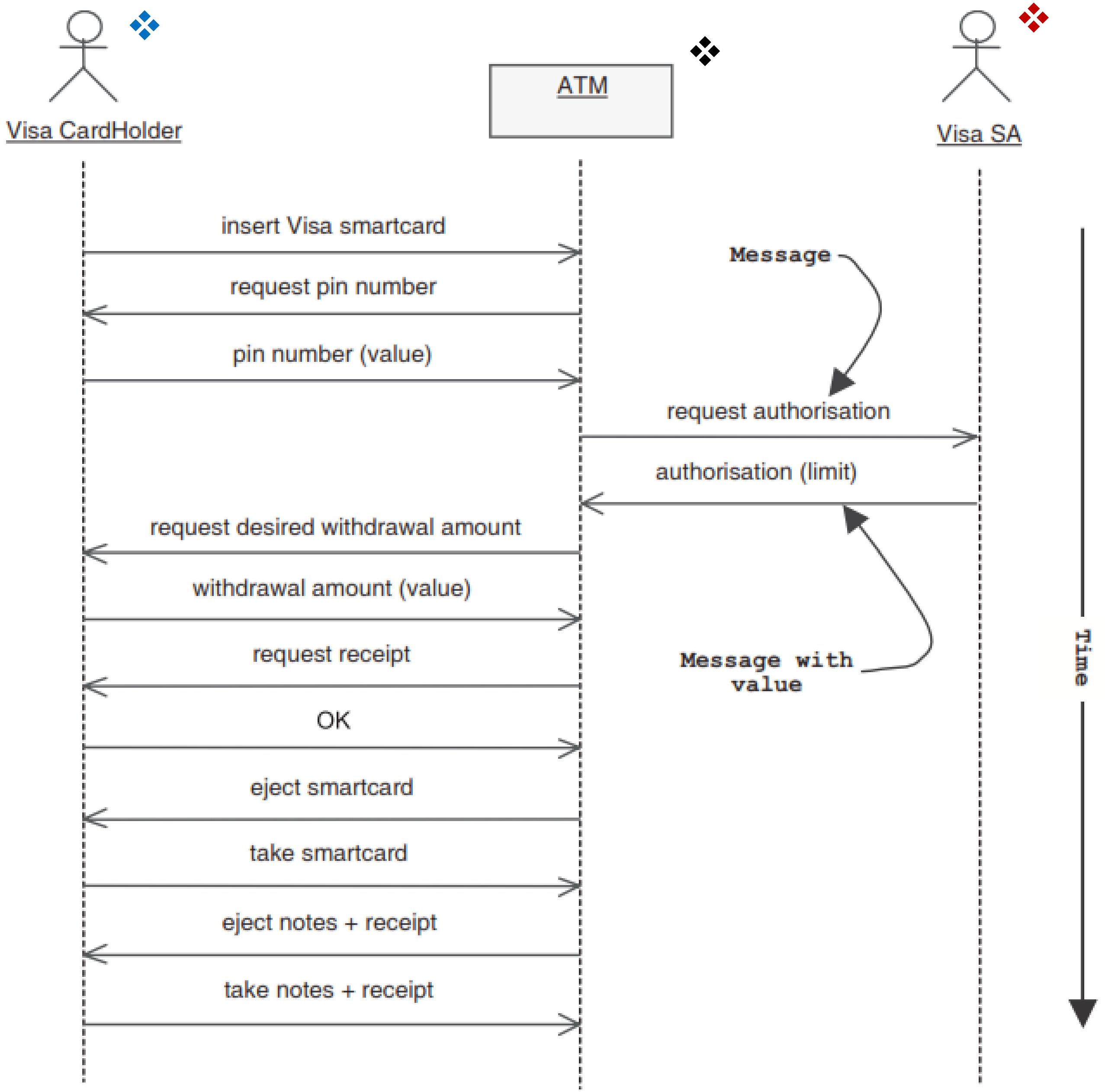
Preconditions
Postconditions

Dynamic descriptions of a use case

Case Study – ATM

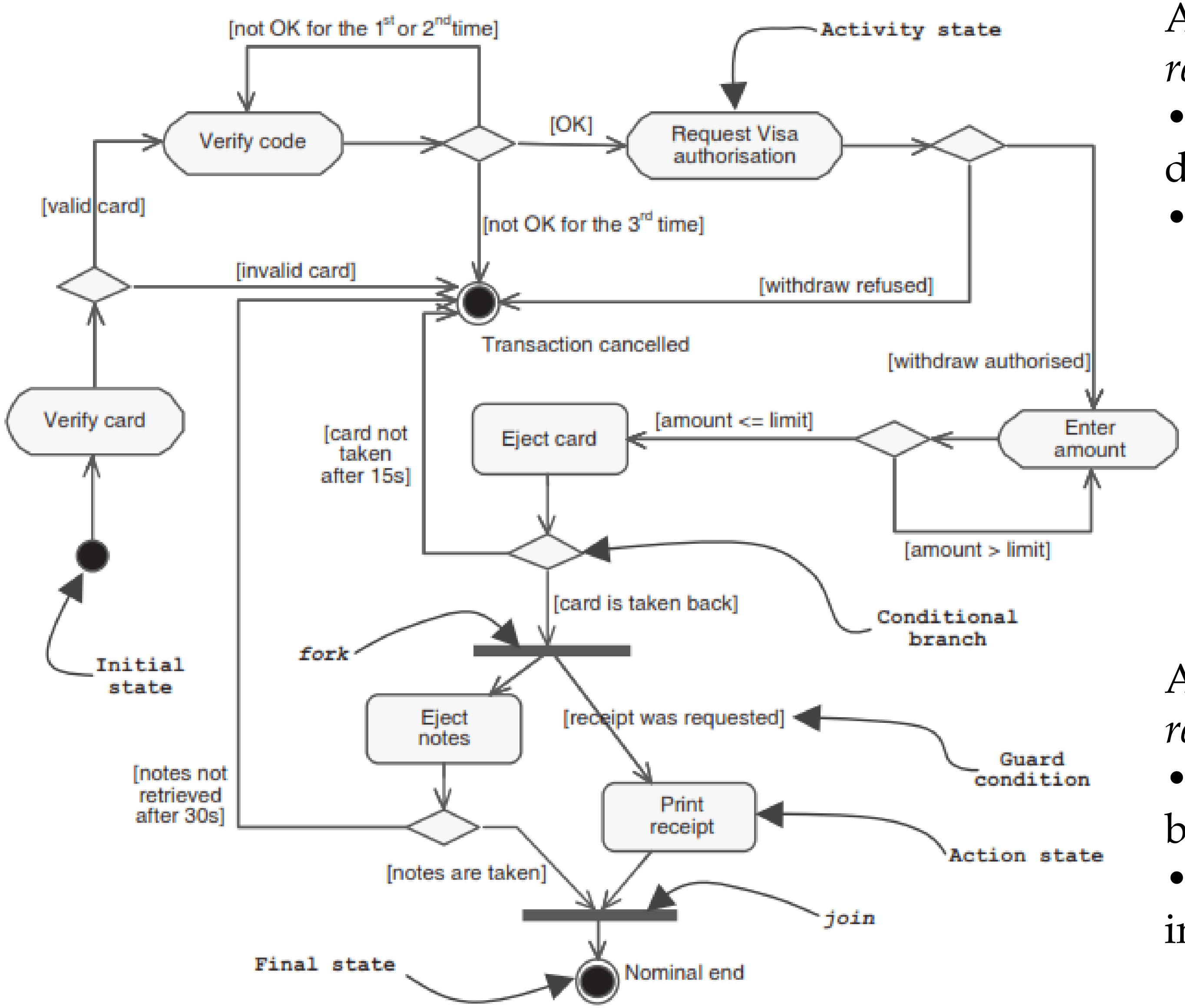
Step #5: Graphical description of use cases

- ❖ primary actor on the left
- ❖ the system in a black box in the middle
- ❖ any secondary actors on the right



Case Study – ATM (Automatic Teller Machine)

Step #5: Graphical description of use cases



An *activity state models the realization of an activity that:*

- is complex and can be broken down into activities or actions,
- can be interrupted by an event.

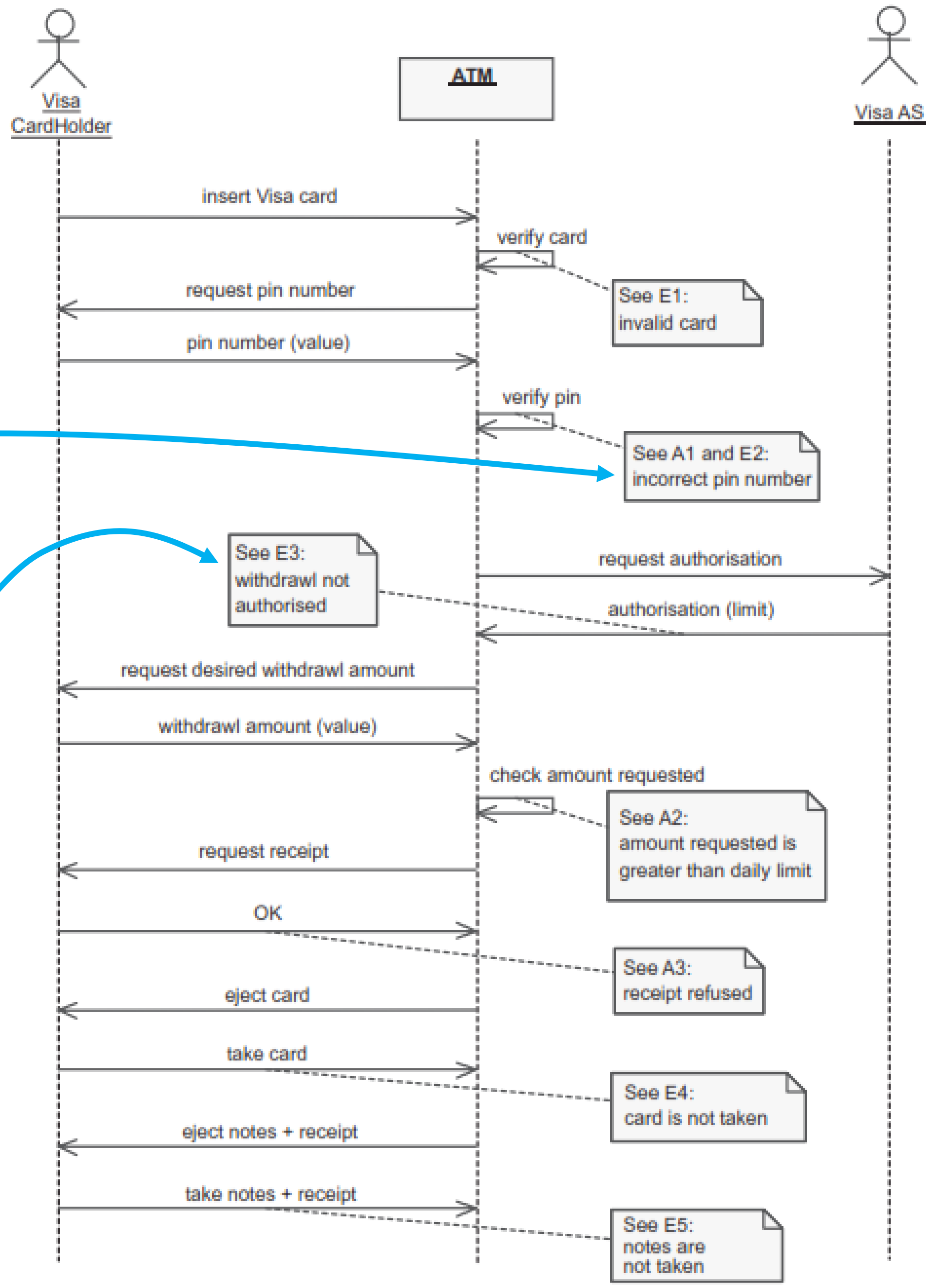
An *action state models the realization of an action that:*

- is simple and cannot be broken down,
- is atomic, which cannot be interrupted.

Case Study – ATM (Automatic Tel

Step #5: Graphical description of use cases

references to “alternative” and error sequences (by means of notes).

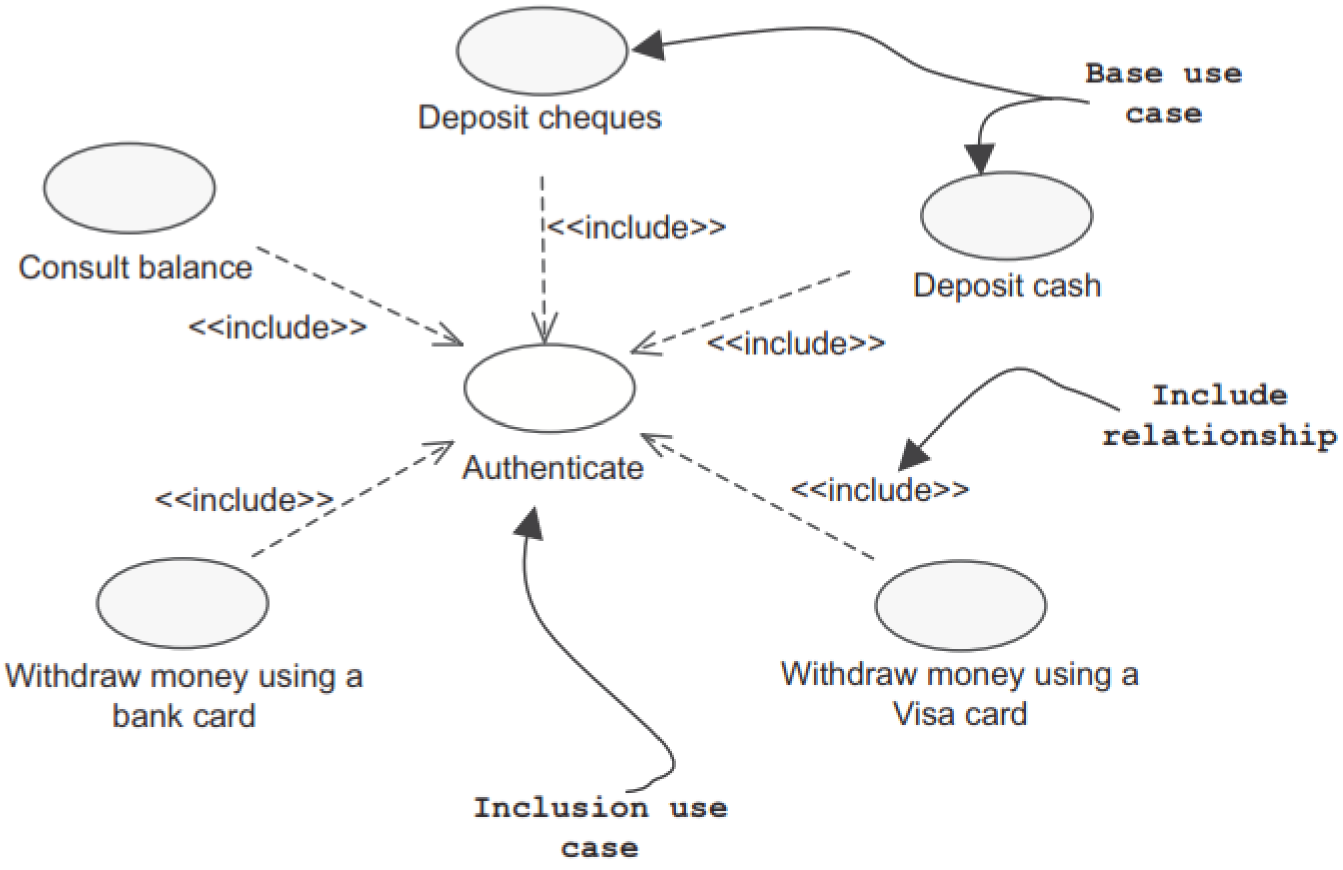


Case Study – ATM (Automatic Teller Machine)

Step #6: Organizing the use cases

With UML, it is actually possible to detail and organize use cases in two 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 – ATM (Automatic Teller Machine)

Step #6: Organizing the use cases

extend: a relationship from an extension use case to a base use case, specifying how the behavior defined for the extension use case augments (subject to conditions specified in the extension) the behavior defined for the base use case.

- The behavior is inserted at the location defined by the extension point in the base use case. The base use case does not depend on performing the behavior of the extension use case.
- Note that the extension use case is **optional** unlike the included use case which is mandatory.
- We use this relationship to separate an optional or rare behavior from the mandatory behavior.

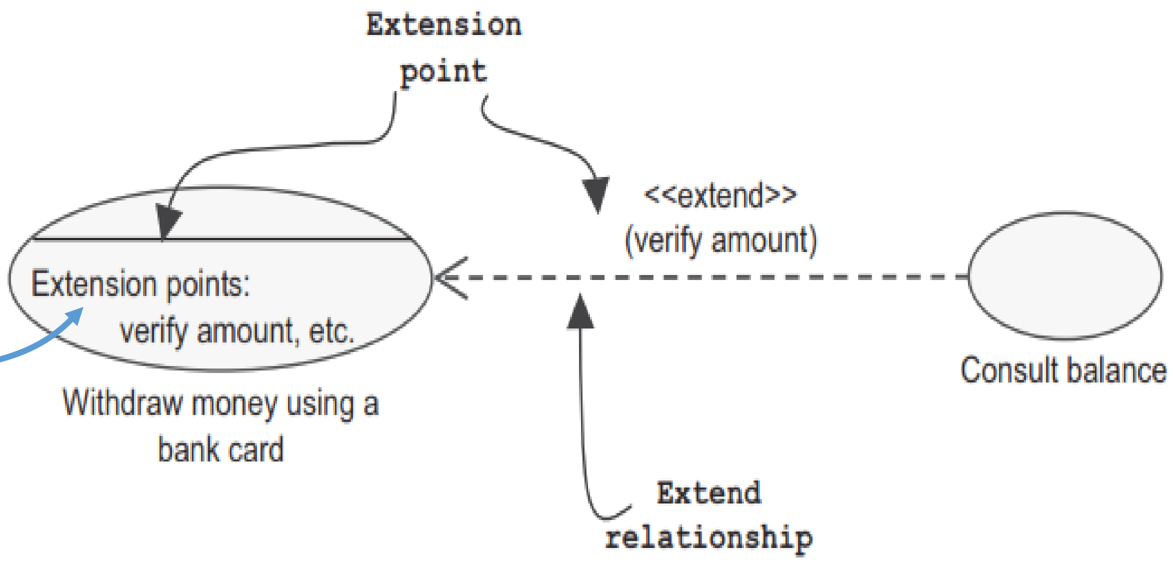
...

7. The VISA authorisation system confirms its agreement and indicates the daily withdrawal limit.
8. The ATM asks the Bank customer to enter the desired withdrawal amount.

   Extension point: Verify amount

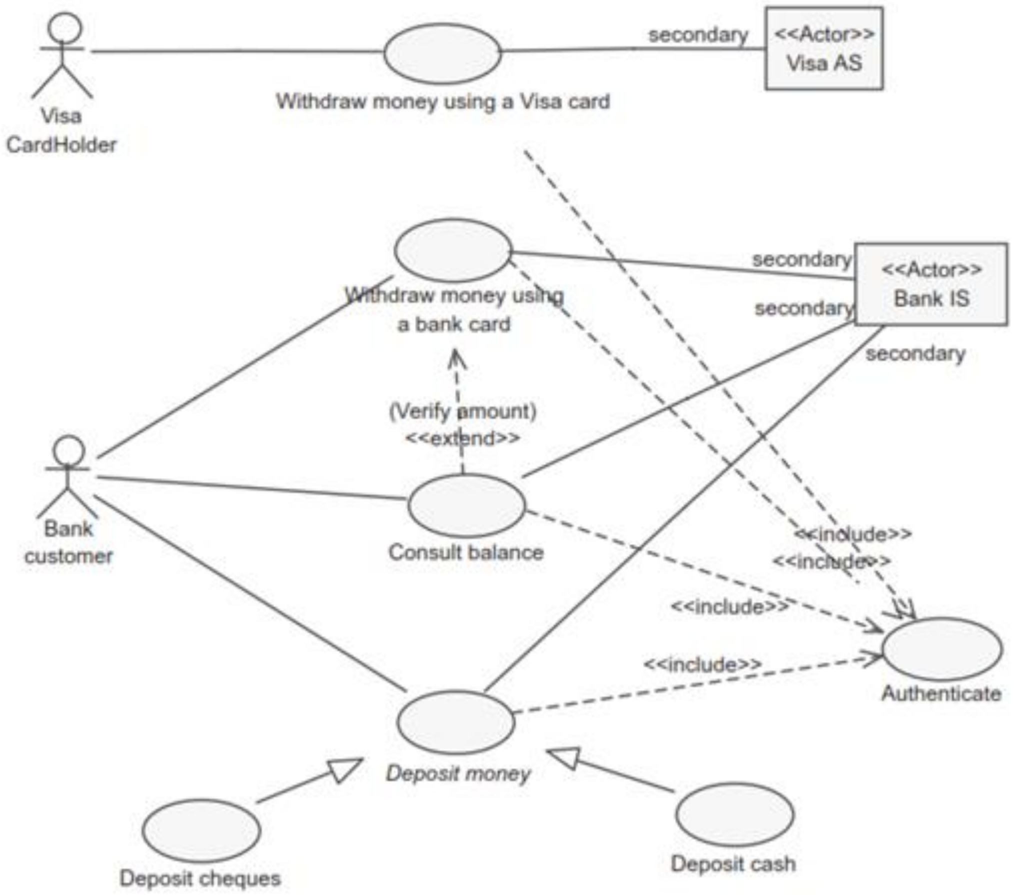
9. The Bank customer enters the desired withdrawal amount.
10. The ATM checks the desired amount against the daily withdrawal limit.

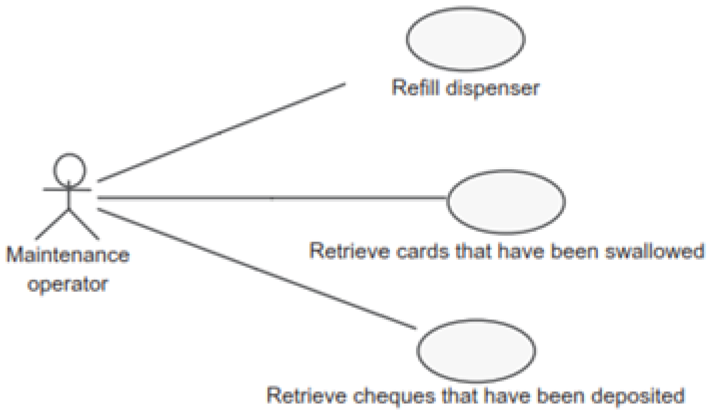
...



Case Study – ATM (Automatic Teller Machine)

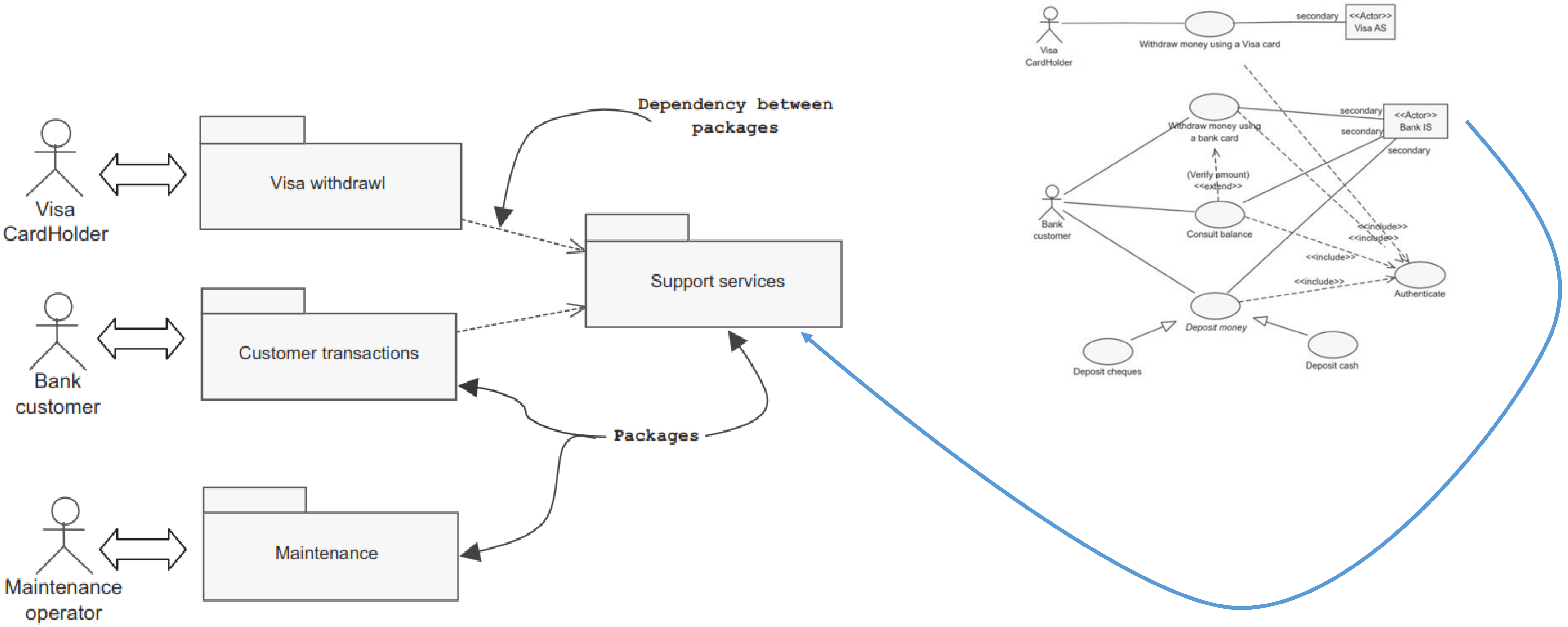
Step #6: Organizing the use cases





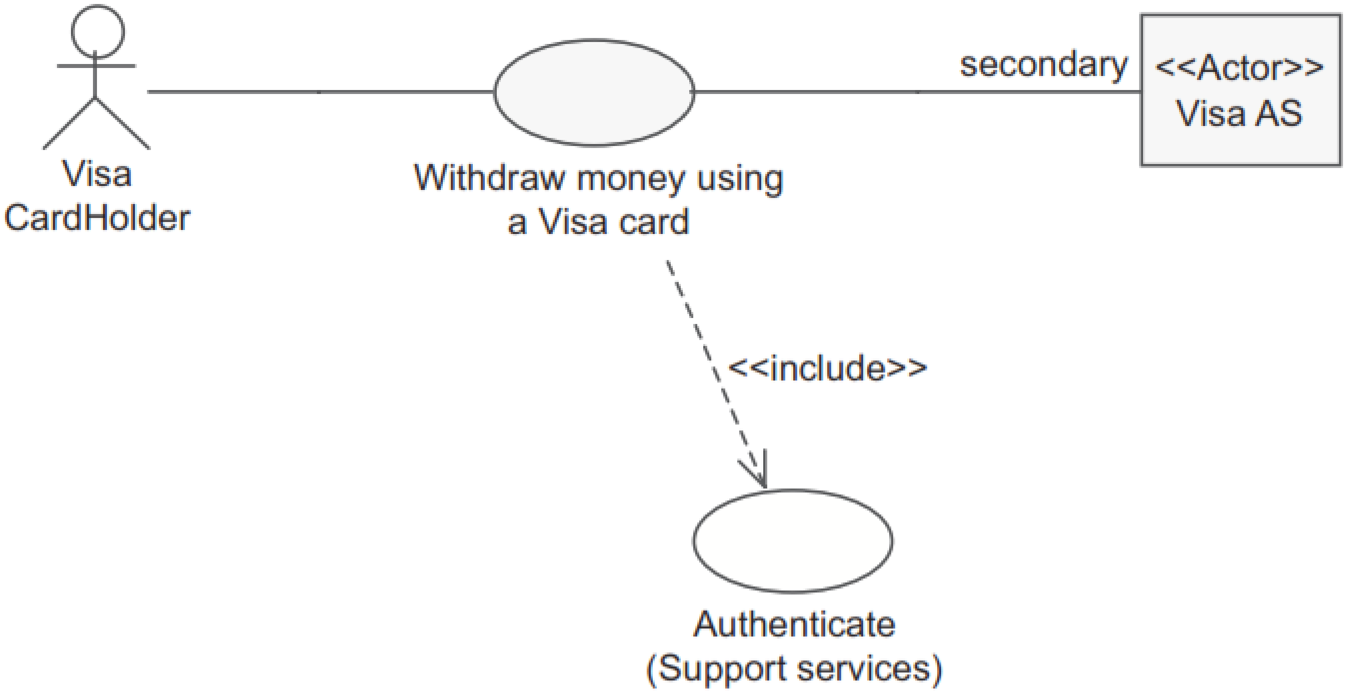
Case Study – ATM (Automatic Teller Machine)

Step #6: Organizing the use cases



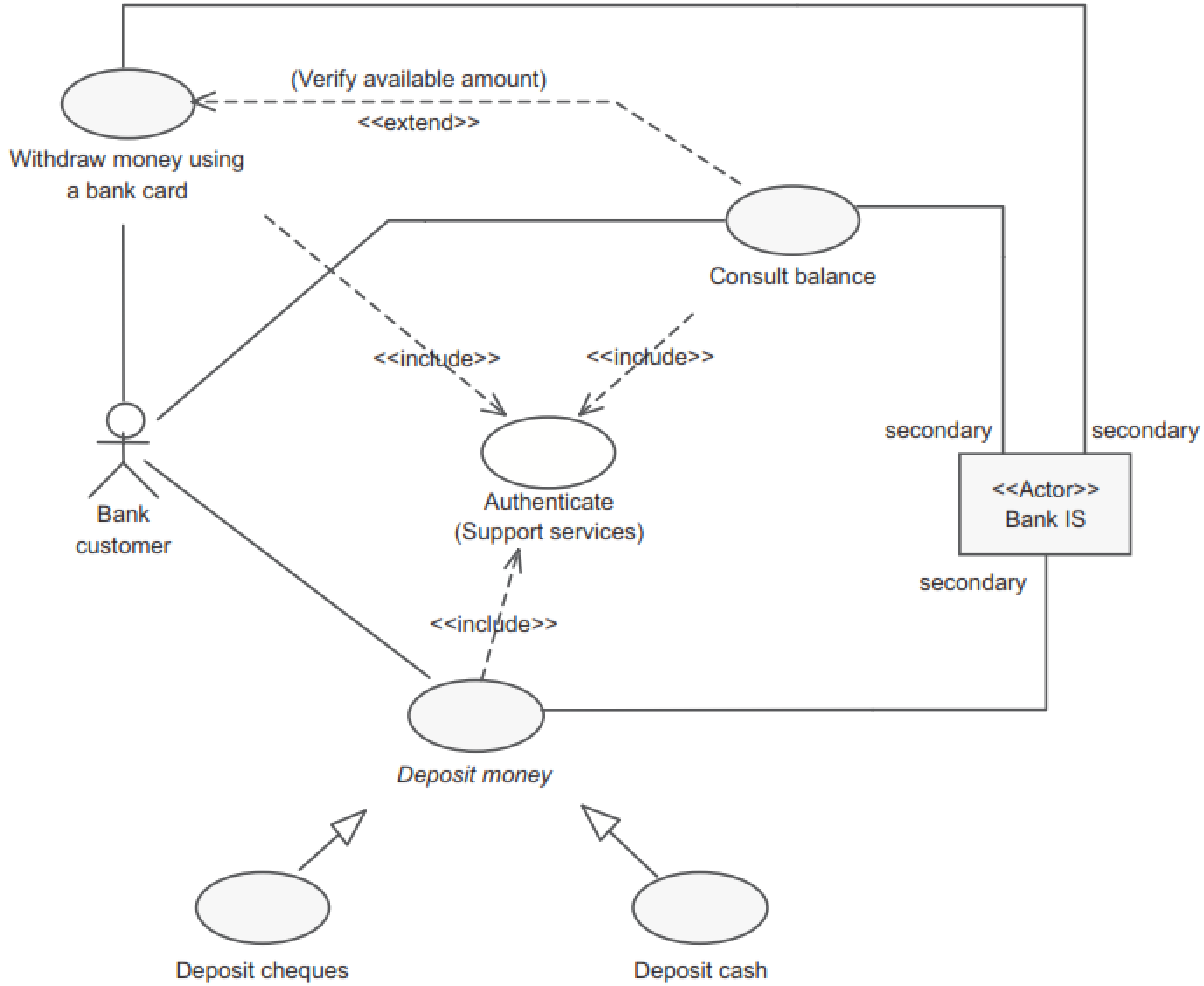
Case Study – ATM (Automatic Teller Machine)

Step #6: Organizing the use cases



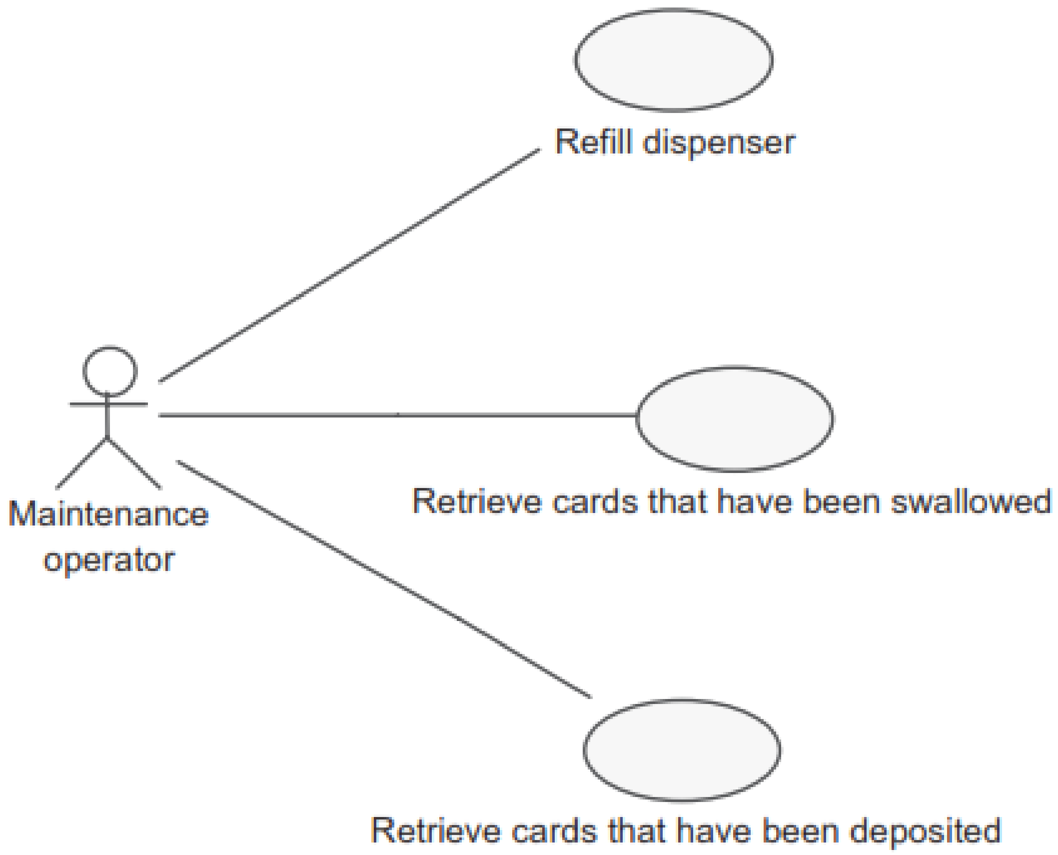
Case Study – ATM (Automatic Teller Machine)

Step #6: Organizing the use cases



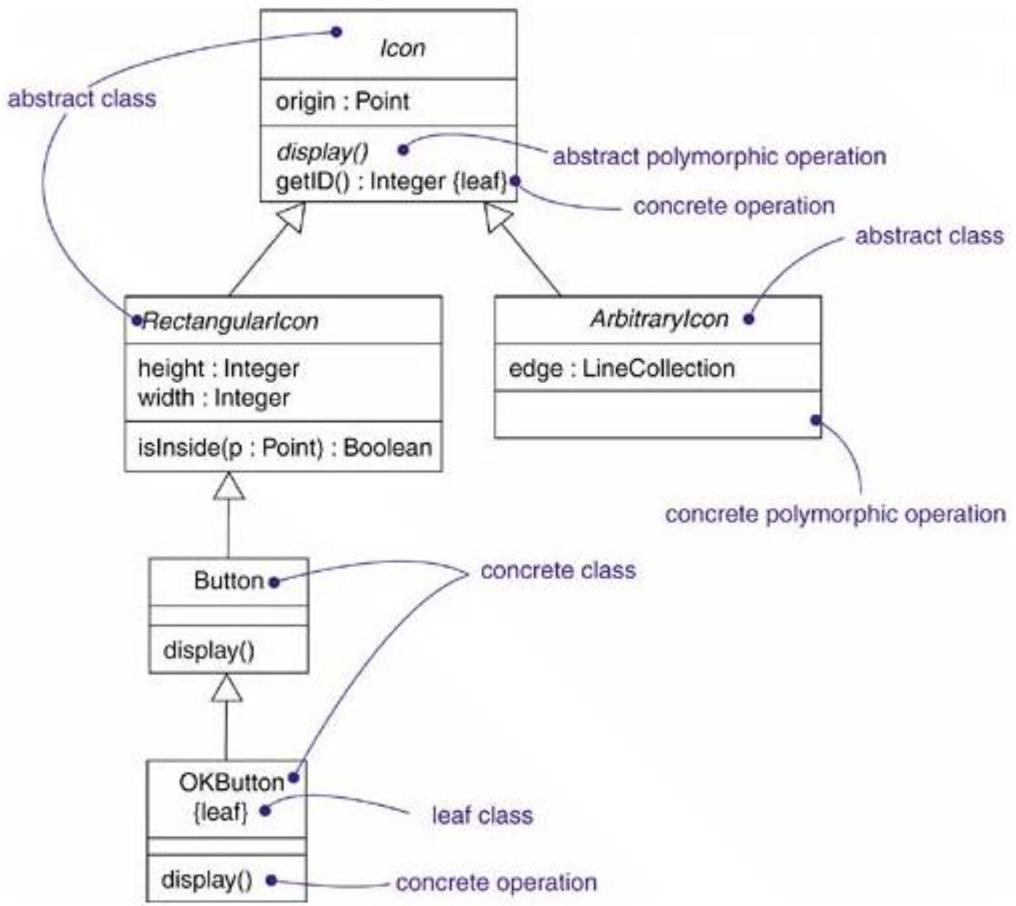
Case Study – ATM (Automatic Teller Machine)

Step #6: Organizing the use cases

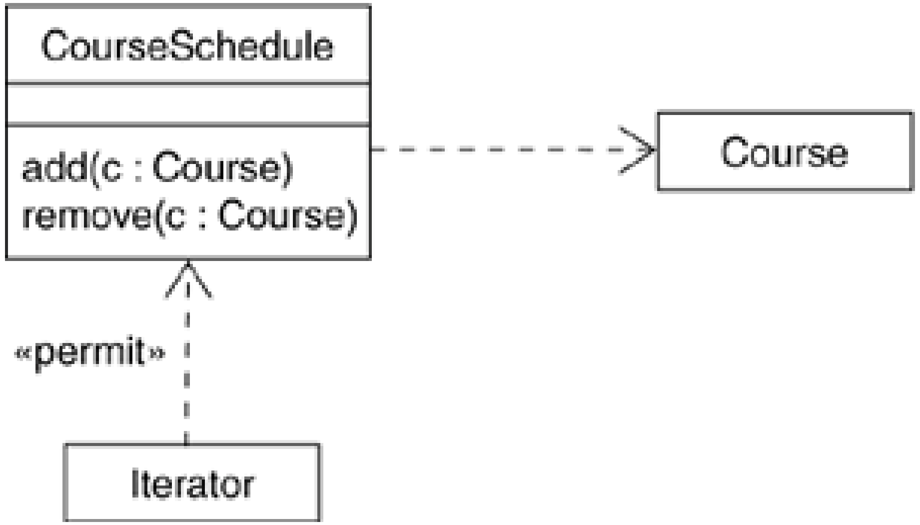


Other Samples

Figure 9-5. Abstract and Concrete Classes and Operations



Other Samples



The dependency from Iterator shows that the Iterator uses the CourseSchedule; the CourseSchedule knows nothing about the Iterator.
The dependency is marked with the stereotype **«permit»**, which is similar to the **friend** statement in C++.

Requirements Gathering Techniques

- Interview (Open or Closed=Structured)
- Observation
  - Direct (Observing live functions in the working space either active or passive)
  - Indirect (media or movie)
- Research / Questionnaire
- Documents Analysis
- Reverse Engineering (black-box or white-box)
- Prototyping
- Brainstorming
- Focus Group
- Interface Identification (User/System/Hardware)
- Storyboarding / Storytelling
- Role Playing
- Requirements Workshops