UML in Practice

The Art of Modeling Software Systems Demonstrated through Worked Examples and Solutions

Pascal Roques

JOHN WILEY & SONS, LTD

1

Case study: automatic teller machine

Aims of the chapter

By means of the first case study, this chapter will allow us to illustrate the main difficulties step by step, which are connected to implementing the technique of use cases.

Once we have identified the actors that interact with the system, we will develop our first UML model at a system level, in order to be able to establish precisely the boundaries of the system.

We will then learn how to identify use cases, and how to construct use case diagrams linking actors and use cases. Then we will see how to specify the functional view by explaining in detail the different ways in which actors can use the system. For this goal, we will learn to write textual descriptions as well as to draw complementary UML diagrams (such as sequence or activity diagrams).

Elements involved

- Actor
- Static context diagram
- Use case
- Use case diagram
- Primary actor, secondary actor
- Textual description of a use case
- Scenario, sequence
- System sequence diagram
- Activity diagram

- Inclusion, extension and generalisation of use cases
- Packaging use cases.

Case study 1 – Problem statement

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 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.

Do not forget either that:

3. All transactions are made secure.
4. It is sometimes necessary to refill the dispenser, etc.

From these four sentences, we will work through the following activities:

- 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,
- Organise and structure the use cases.

Watch out: the preceding problem statement is deliberately incomplete and imprecise, just as it is in real projects!

Note also that the problem and its solution are based on French banking systems and the use of smartcards: the system you actually use in your country may be significantly different! It is not very important. What is important is the way of thinking to solve this functional problem as well as the UML concepts and diagrams that we use.

1.1 Step 1 – Identifying the actors of the ATM

First, we will identify the actors of the ATM system.

An actor is a construct employed in use cases that define a role that a user or any other system plays when interacting with the system under consideration. It is a type of entity that interacts, but which is itself external to the subject. Actors may represent human users, external hardware, or other subjects. An actor does not necessarily represent a specific physical entity. For instance, a single physical entity may play the role of several different actors and, conversely, a given actor may be played by multiple physical entities.[3]

** 1.1 Identify the main actors of the ATM.

Answer 1.1

What are the external entities that interact directly with the ATM?

Let's look at each of the sentences of the exposition in turn.

Sentence 1 allows us to identify an obvious initial actor straight away: every "holder of a smartcard". He or she will be able to use the ATM to withdraw money using his or her smartcard.

However, be careful: the card reader and cash dispenser constitute part of the ATM. They can therefore not be considered as actors! You can note down that the identification of actors requires the boundary between the system being studied and its environment to be set out exactly. If we restrict the study to the control/ command system of physical elements of the ATM, the card reader and cash dispenser then become actors.

Another trap: is the smartcard itself an actor? The card is certainly external to the ATM, and it interacts with it... Yet, we do not recommend that you list it as an actor, as we are putting into practice the following principle: eliminate "physical" actors as much as possible to the advantage of "logical" actors. The actor is the who or what that benefits from using the system. It is the card holder who withdraws money to spend it, not the card itself!

Sentence 2 identifies additional services that are only offered to bank customers who hold a smartcard from this bank. This is therefore a different profile from the previous one, which we will realise by a second actor called *Bank customer*.

Sentence 3 encourages us to take into account the fact that all transactions are made secure. But who makes them secure? There are therefore other external entities, which play the role of authorisation system and with which the ATM

3. From the OMG document: "Unified Modeling Language: Superstructure (version 2.0)".

communicates directly. An interview with the domain expert[4] is necessary to allow us to identify two different actors:

- the Visa authorisation system (VISA AS) for withdrawal transactions carried out using a Visa smartcard (we restrict the ATM to Visa smartcards for reasons of simplification);

- the information system of the bank (Bank IS) to authorise all transactions carried out by a customer using his or her bank smartcard, but also to access the account balance.

Finally, sentence 4 reminds us that an ATM also requires maintenance work, such as refilling the dispenser with bank notes, retrieving cards that have been swallowed, etc. These maintenance tasks are carried out by a new actor, which – to simplify matters – we will call *Maintenance operator*.

Graphical representations of an actor

The standard graphical representation of the actor in UML is the icon called *stick man* with the name of the actor below the drawing. It is also possible to show an actor as a class rectangle with the <<`actor`>> keyword. A third representation (halfway between the first two) is also possible, as indicated below.

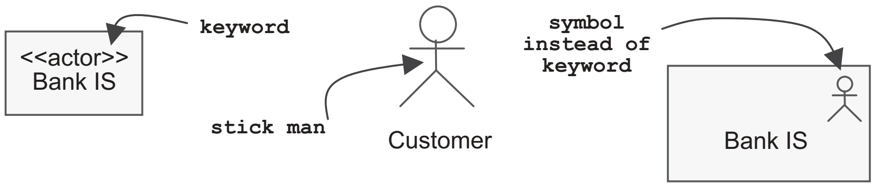

Figure 1.1 Possible graphical representations of an actor

A good piece of advice consists in using the graphical form of the *stick man* for human actors and that of the first rectangular representation for connected systems.

---

4. Remember that the domain refers to French banking systems, which may explain differences with your own knowledge and experience.

Rather than simply depicting the list of actors as in the previous figure, which does not provide any additional information with regard to a textual list, we can draw a diagram that we will call *static context diagram*. To do this, simply use 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.

Even though this is not a traditional UML diagram, we have found this kind of "context diagram" very useful in our practical experience.

** 1.2 Map out the static context diagram of the ATM.

Answer 1.2

The ATM is fundamentally a single user system: at any moment, there is only one instance of each actor (at the most) connected to the system.

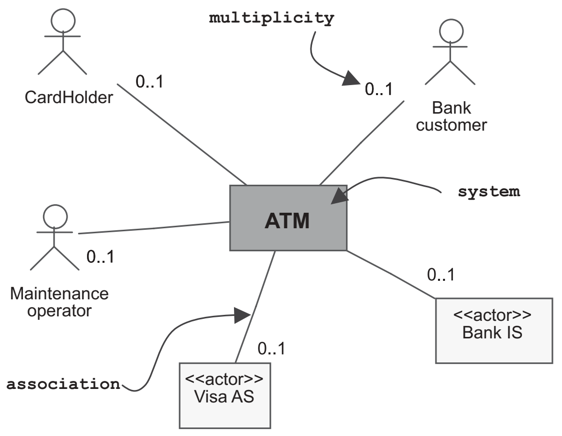

Figure 1.2 Static context diagram of the ATM

We should really add a graphical note to indicate that the human actors, *Bank customer* and *CardHolder* are, furthermore, mutually exclusive, which is not implicit according to the multiplicities of the associations.

Another solution, which is a little more developed, consists in considering *Bank customer* as a specialisation of *CardHolder*, as illustrated in the following figure. The aforementioned problem of exclusivity is therefore solved by adding an extra detail to the diagram.

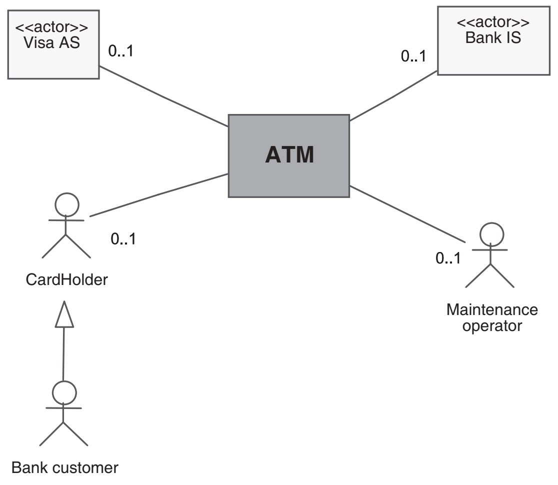

Figure 1.3 A more developed version of the static context diagram of the ATM

1.2 Step 2 – Identifying use cases

We are now going to identify the 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.[5]

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.

**

1.3 Prepare a preliminary list of use cases of the ATM, in order of actor.

Answer 1.3

Let's take the five actors one by one and list the different ways in which they can use the ATM:
CardHolder:

- Withdraw money.

5. From the OMG document: "Unified Modeling Language: Superstructure (version 2.0)".

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.

Visa authorisation system (AS):

- None.

Bank information system (IS):

- None.

Primary or secondary actor

Contrary to what we might believe, all actors do not necessarily use the system! We call the one for whom the use case produces an observable result the *primary* actor. In contrast, *secondary* actors constitute the other participants of the use case.[6] Secondary actors are requested for additional information; they can only consult or inform the system when the use case is being executed.

This is exactly the case of the two “non-human” actors in our example: the *Visa AS* and the *Bank IS* are only requested by the ATM within the context of realising certain use cases. However, they themselves do not have their own way of using the ATM.

6. In his excellent book, *Writing Effective Use Cases* (Addison-Wesley, 2001), A. Cockburn defines similarly *supporting actors*: “A supporting actor in a use case is an external actor that provides a service to the system under design.”

1.3 Step 3 – Creating use case diagrams

We are now going to give concrete expression to our identification of use cases by realising UML diagrams, aptly called use case diagrams. A use case diagram shows the relationships among actors and the subject (system), and use cases.

We can easily obtain a preliminary diagram by copying out the previous answer on a diagram that shows the use cases (ellipses) inside the ATM system (box) and linked by associations (lines) to their primary actors (the "stick man" icon).

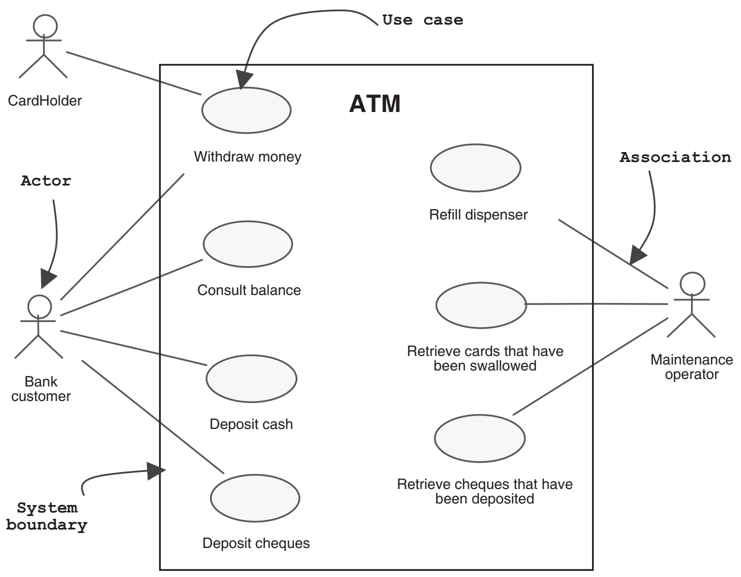

Figure 1.4 Preliminary use case diagram of the ATM

***

1.4 Propose another, more sophisticated version of this preliminary use case diagram.

Answer 1.4

The *Withdraw money* use case has two possible primary actors (but they cannot be simultaneous). Another way to express this notion is to consider the *Bank customer* actor as a specialisation (in the sense of the inheritance relationship) of the more general *CardHolder* actor. A bank customer is actually a particular card holder who has all the privileges of the latter, as well as others that are specific to him or her as a customer.

UML enables the depiction of a generalisation/specialisation relationship between actors, as indicated on the diagram below.

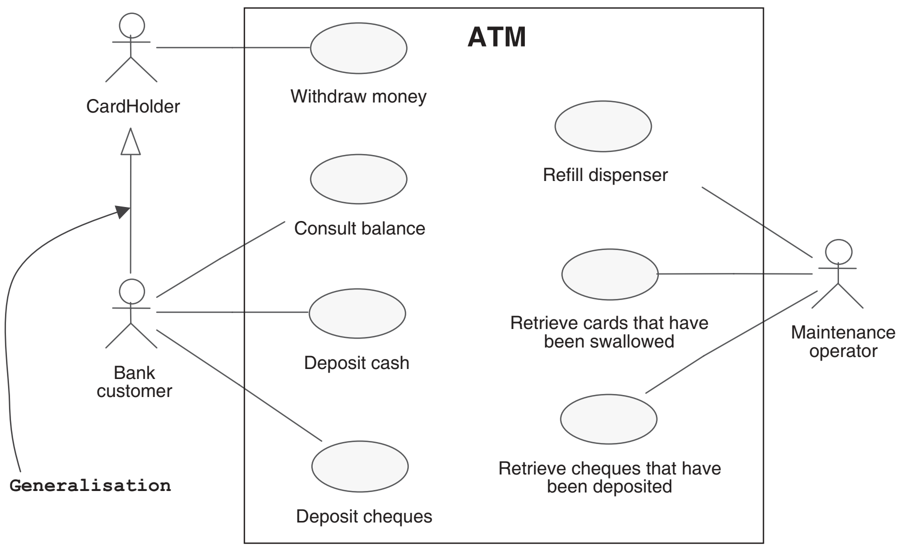

Figure 1.5 A more sophisticated version of the preliminary use case diagram

However, the significance of this generalisation relationship is not evident in our example. Certainly, it enables the association between the *Bank customer* actor and the *Withdraw money* use case to be removed, which is now inherited from the *CardHolder* actor, but on the other hand, it adds the symbol for generalisation between the two actors... Moreover, we will see in the following paragraph that the requested secondary actors are not the same in the case of the CardHolder and in that of the bank customer.

We will therefore not use this solution and, to reinforce this choice, we will rename the primary actor *Visa CardHolder*, to clarify matters a little more.

We now have to add the secondary actors in order to complete the use case diagram. To do this, we will simply make these actors appear with additional associations towards the existing use case.

Graphical precisions on the use case diagram

As far as we are concerned, we recommend that you adopt the following conventions in order to improve the informative content of these diagrams:

- by default, the role of an actor is "primary"; if this is not the case, indicate explicitly that the role is "secondary" on the association to the side of the actor;
- as far as possible, place the primary actors to the left of the use cases and the secondary actors to the right.

**

1.5 Complete the preliminary use case diagram by adding the secondary actors. To simplify matters, leave out the maintenance operator for the time being.

Answer 1.5

For all use cases appropriate for the bank customer, you must explicitly bring in *Bank IS* as a secondary actor.

But a problem arises for the shared use case, *Withdraw money*. Indeed, if the primary actor is a Visa card holder, the *Visa AS* must be called on (which will then be responsible for contacting the IS of the holder's bank); whereas the ATM will contact the *Bank IS* directly if it concerns a bank customer.[7]

One solution consists in adding an association with each of the two non-human actors. This simplistic modelling does not make it clear to the reader of the diagram that the actors are selectively participating two by two and not all together.

7. Remember that the domain refers to French banking systems, which may explain differences with your knowledge and experience.

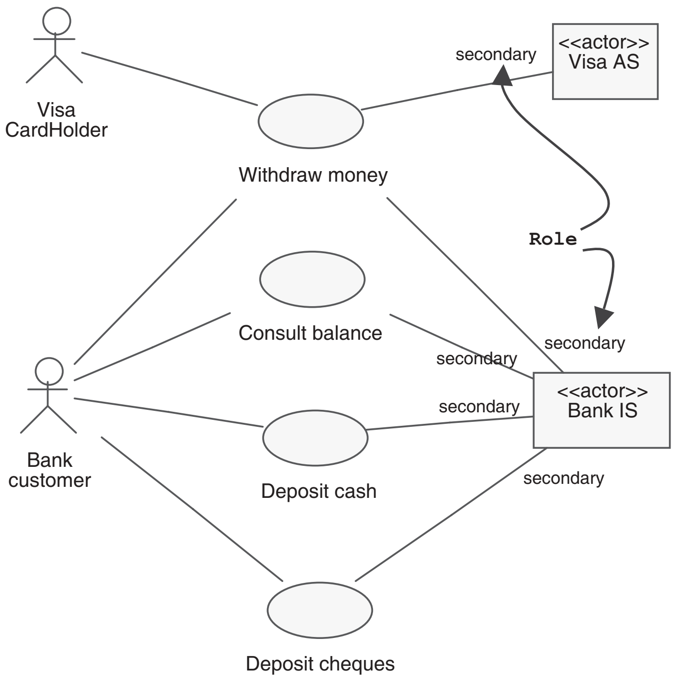

Figure 1.6 Simple version of the completed use case diagram

Another solution would be to distinguish two use cases for the withdrawal of money: *Withdraw money using a Visa card* and *Withdraw money using a bank card*. This more precise, yet more cumbersome, modelling is easier for the reader of the diagram to grasp. Furthermore, it clearly tells against the use of generalisation between actors, which was mentioned beforehand. Indeed, the distinction between the two use cases is contradictory with the attempt at inheritance of the unique *Withdraw money* case, which had been viewed more highly, while the secondary actors had not yet been added. We will keep this second solution for the follow-up to the exercise.

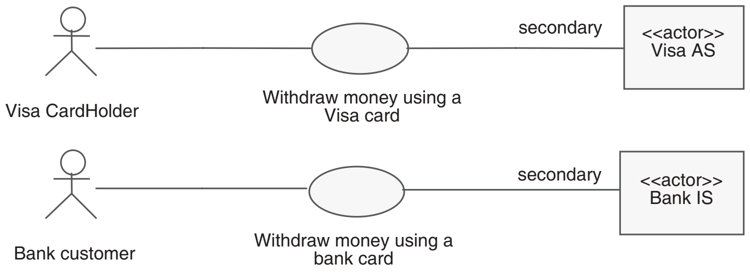

Figure 1.7 Fragment of the more precise version of the completed use case diagram

We will note that the *Bank IS* is not a direct actor of the *Withdraw money using a Visa card* use case, as we are considering that the *Visa AS* is taking upon itself to contact it, outside of reach of the ATM system. Obviously, if the bank issue money to a Visa customer, they need to claim this money back from Visa. We assume this is out of scope.

1.4 Step 4 – Textual description of use cases

Once the use cases have been identified, you then have to describe them!

In order to explain the dynamics of a use case in detail, the most obvious way of going about it involves textually compiling a list of all the interactions between the actors and the system. The use case must have a clearly identifiable beginning and end. The possible variants must also be specified, such as the main success scenario, alternative sequences, error sequences, whilst simultaneously trying to arrange the descriptions in a sequential order in order to improve their readability.

Scenarios and 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.[8]

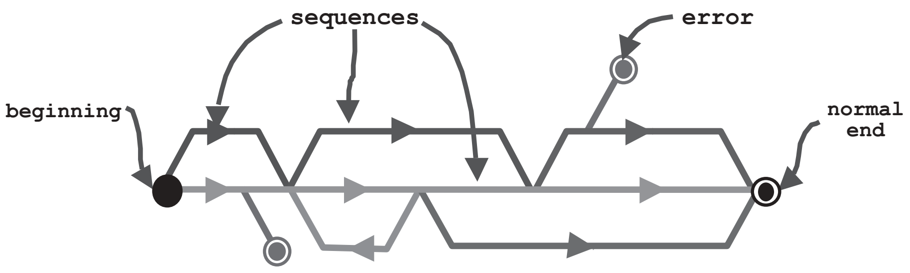

Figure 1.8 Representation of the scenarios of a use case

8. From the OMG document: "Unified Modeling Language: Superstructure (version 2.0)".

The textual description record of a use case is not standardised by UML.[9] For our part, we recommend the following structuring:

Identification summary (mandatory)

- includes title, summary, creation and modification dates, version, person in charge, actors...

Flow of events (mandatory)

- describes the main success scenario,[10] the alternative and error sequences,[11] as well as the preconditions and the postconditions.

UI requirements (optional)

- possibly adds graphical user interface constraints (required look and feel). Screen copies, indeed a disposable model, are greatly appreciated.

Non-functional constraints (optional)

- may possibly add the following information: frequency, availability, accuracy, integrity, confidentiality, performance, concurrency, etc.

** 1.6 Describe the mandatory part of the *withdraw money using a visa card* use case.

Answer 1.6

Identification summary

Title: Withdraw money using a Visa card

Summary: this use case allows a Visa card holder, who is not a customer of the bank, to withdraw money if his or her daily limit allows it.

9. You can find use case templates on the Web, for instance on www.usecases.org.
10. The main success scenario is also known as "basic flow of events" or "normal path".
11. The distinction we make is that with an alternative scenario, the primary actor achieves his or her goal, even though with an error one, the actor's goal is not achieved and the use case fails.

Actors: Visa CardHolder (primary), Visa AS (secondary).

Creation date: 02/03/02 **Date of update:** 08/19/03

Version: 2.2 **Person in charge:** Pascal Roques

Flow of events

Preconditions:

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

Main success scenario:

1. The Visa CardHolder inserts his or her smartcard in the ATM's card reader.
2. The ATM verifies that the card that has been inserted is indeed a smartcard.
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 smartcard.[12]
6. The ATM requests an authorisation from the VISA authorisation system.
7. The VISA authorisation system confirms its agreement and indicates the daily withdrawal limit.
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 withdrawal limit.
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 banknotes and a receipt.
16. The Visa CardHolder takes the banknotes and the receipt.

12. Remember that the use case assumes smartcards, which contain the PIN, contrarily to "ordinary" cards with a magnetic stripe on the back as in North America.

Another possible presentation[13] consists in separating the actions of the actors and those of the system into two columns as follows:

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.	

"Alternative" sequences:

A1: temporarily incorrect pin number
The A1 sequence starts at point 5 of the main success scenario.

6. The ATM informs the CardHolder that the pin is incorrect for the first or second time.
7. The ATM records the failure on the smartcard.

13. This presentation option was recommended by C. Larman in the first version of his book: *Applying UML and Patterns*, Prentice Hall, 1997.

The scenario goes back to point 3.

A2: the amount requested is greater than the daily withdrawal limit
The A2 sequence starts at point 10 of the main success scenario.

11. The ATM informs the CardHolder that the amount requested is greater than the daily withdrawal limit.

The scenario goes back to point 8.

A3: the Visa CardHolder does not want a receipt
The A3 sequence starts at point 11 of the main success scenario.

12. The Visa CardHolder declines the offer of a receipt.
13. The ATM returns the smartcard to the Visa CardHolder.
14. The Visa CardHolder takes his or her smartcard.
15. The ATM issues the banknotes.
16. The Visa CardHolder takes the banknotes.

Error sequences:

E1: invalid card
The E1 sequence starts at point 2 of the main success scenario.

3. The ATM informs the Visa CardHolder that the smartcard is not valid (unreadable, expired, etc.) and confiscates it; the use case fails.

E2: conclusively incorrect pin number
The E2 sequence starts at point 5 of the main success scenario.

6. The ATM informs the Visa CardHolder that the pin is incorrect for the third time.
7. The ATM confiscates the smartcard.
8. The VISA authorisation system is notified; the use case fails.

E3: unauthorised withdrawal
The E3 sequence starts at point 6 of the main success scenario.

7. The VISA authorisation system forbids any withdrawal.
8. The ATM ejects the smartcard; the use case fails.

E4: the card is not taken back by the holder
The E4 sequence starts at point 13 of the main success scenario.

14. After 15 seconds, the ATM confiscates the smartcard.

15. The VISA authorisation system is notified; the use case fails.

E5: the banknotes are not taken by the holder
The E5 sequence starts at point 15 of the main success scenario.

16. After 30 seconds, the ATM takes back the banknotes.

17. The VISA authorisation system is informed; the use case fails

Postconditions:

- The cashbox of the ATM contains fewer notes than it did at the start of the use case (the number of notes missing depends on the withdrawal amount).

---

* 1.7 Complete the description of the *withdraw money using a visa card* use case with the two optional paragraphs. Assume for instance that the new system must run on existing ATM hardware.

Answer 1.7

UI requirements

The input/output mechanisms available to the Visa CardHolder must be:

- A smartcard reader.
- A numerical keyboard (to enter his or her pin number), with "enter", "correct" and "cancel" keys.
- A screen to display any messages from the ATM.
- Keys around the screen so that the card holder can select a withdrawal amount from the amounts that are offered.
- A note dispenser.
- A receipt dispenser.

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}.

1.5 Step 5 – Graphical description of use cases

The textual description is essential for the documentation of use cases, as it alone enables ease of communication with users, as well as agreeing on domain terminology that is used.

However, the text also has its disadvantages as it difficult to show how the sequences follow one another, or at what moment the secondary actors are requested. Besides, keeping a record of changes often turns out to be rather tiresome. It is therefore recommended to complete the textual description with one or more dynamic UML diagrams.

14. 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.

Dynamic descriptions of a use case

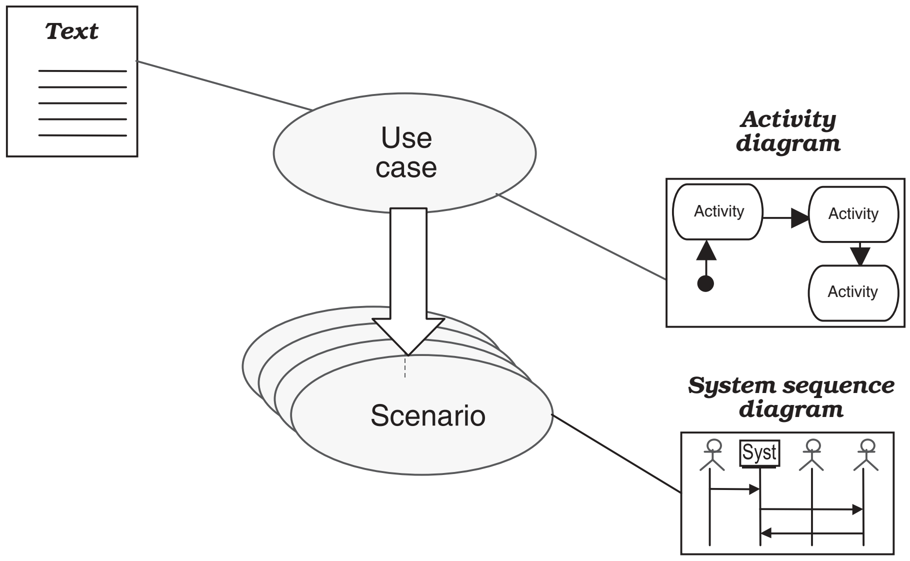

Figure 1.9 UML diagrams that we recommmend for completing the description of a use case

- For use cases, we recommend the *activity diagram*, as users find it far easier to understand since it resembles a traditional diagram. However, the *state diagram* may be useful for use cases that are very interactive.

- For certain scenarios, the *sequence diagram* works well. We recommend that you present it by showing the primary actor on the left, then an object representing the system in a black box, and finally, any secondary actors that may be requested during the scenario on the right of the system. We will use the title *system sequence diagram* as proposed by Larman.[15]

15. Refer to *Applying UML and Patterns* (2nd Edition), Prentice-Hall, 2001.

*

1.8 Create a system sequence diagram that describes the main success scenario of the *Withdraw money using a Visa card* use case.

Answer 1.8

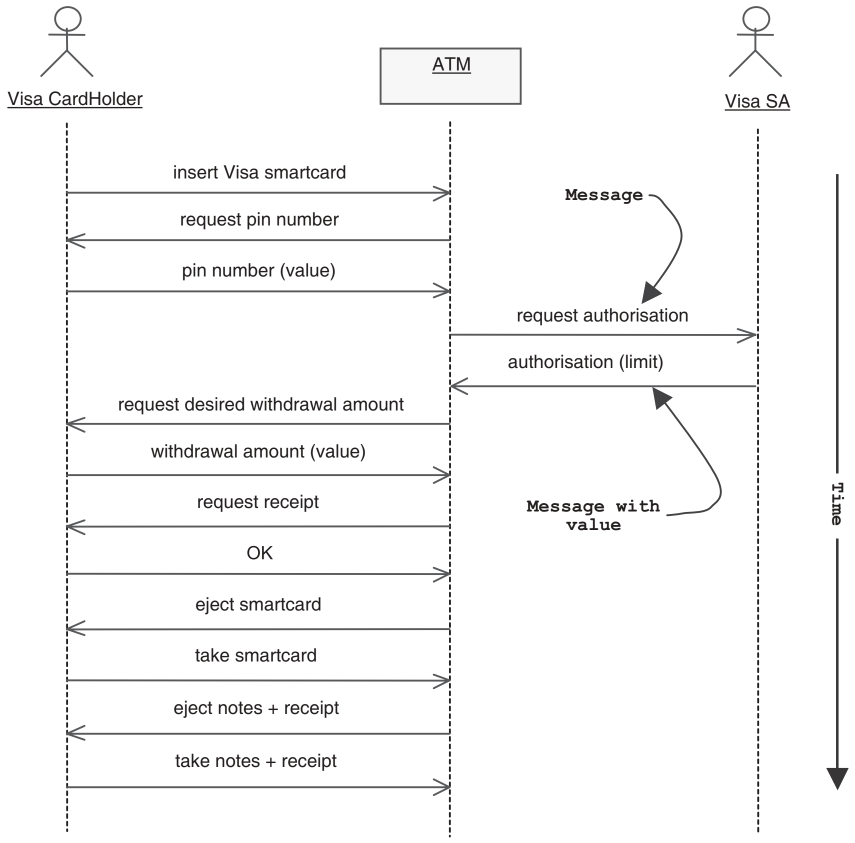

Figure 1.10 System sequence diagram of the "Withdraw money using a Visa card" main success scenario

All you need to do is copy the interactions quoted in the textual scenario of answer 1.6 into a sequence diagram by following the graphical conventions listed above:

- the primary actor, Visa CardHolder, on the left,

- an object representing the ATM system as a whole in the middle,
- the secondary actor, Visa AS, to the right of the ATM.

Unlike the previous sequence diagram that only describes the main success scenario, the activity diagram can represent all the activities that are carried out by the system, with all the conditional branches and all the possible loops.

The activity diagram is essentially a flowchart, showing flow of control from activity to activity. The transitions are triggered at the end of activities or actions; steps can be carried out in parallel or in sequence.

Activity state or action state

An *activity state* models the realisation 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 realisation of an action that:

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

*** 1.9 Construct an activity diagram that describes the dynamics of the *withdraw money using a visa card* use case.

Answer 1.9

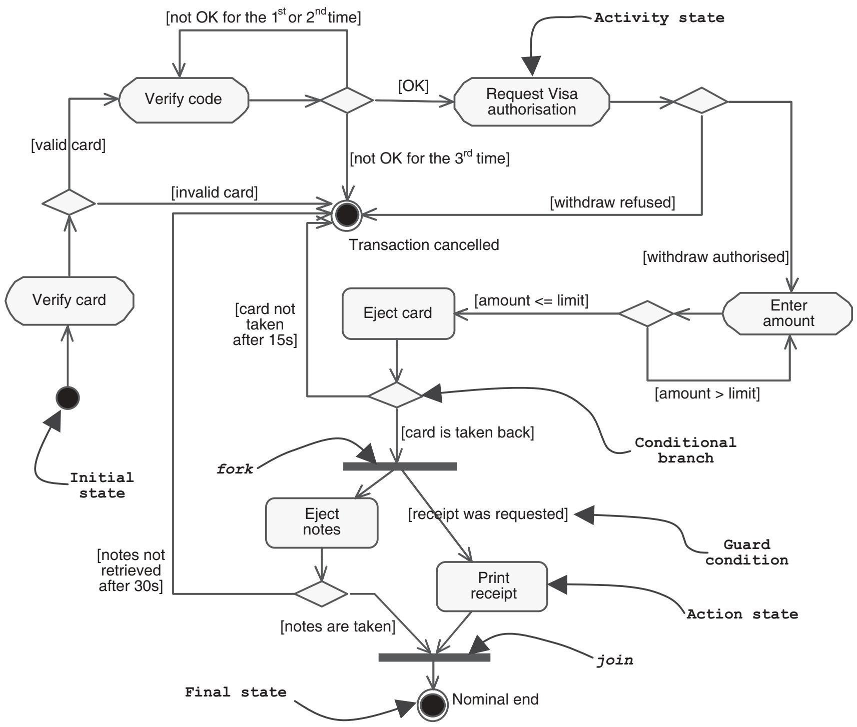

Figure 1.11 Activity diagram of Withdraw money using a Visa card

Note that the activity diagram differs slightly from the text: it omits the step to ask if a receipt is wanted, as we did not want to clutter the diagram. But the result of the step is nonetheless taken into account by the guard condition labelled "receipt was requested".

Additions to the system sequence diagram

A possibility that meets halfway consists in expanding the system sequence diagram of the nominal scenario in order to introduce the following:

- the main internal activities of the system (by means of messages that it sends to itself),
- references to "alternative" and error sequences (by means of notes).

This often results in a diagram that is less complex to read than an activity diagram, as there are fewer symbols, but it nevertheless retains an informative content for the specialist.

* *

1.10 Expand the system sequence diagram that describes the nominal scenario of the *Withdraw money using a visa card* use case.

Answer 1.10

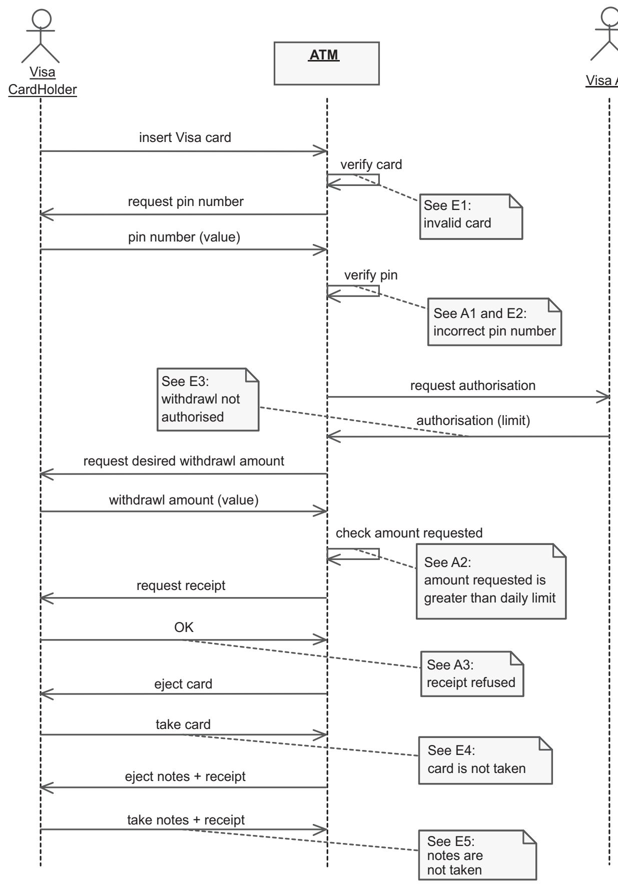

Figure 1.12 Expanded system sequence diagram of the Withdraw money using a Visa card main success scenario

1.6 Step 6 – Organising the use cases

In this final stage, we will refine our diagrams and descriptions.

With UML, it is actually possible to detail and organise use cases in two different and complementary ways:

- by adding include, extend and generalisation relationships between use cases;
- by grouping them into packages to define functional blocks of highest level.

First, let's tackle the *include* relationship: a relationship from a base use case to an inclusion use case, specifying how the behaviour for the base use case contains the behaviour of the inclusion use case. The behaviour is included at the location which is defined in the base use case. The base use case depends on performing the behaviour of the inclusion use case, but not on its structure.[16] We use this relationship to avoid describing the same sequence several times by factorising the shared behaviour in its own use case.

***

1.11 identify a part that the different use cases have in common and factorise it in a new case included in the former.

Answer 1.11

If we examine the textual description of the *Withdraw money using a Visa card* use case in detail, we notice that steps one to five of the main success scenario will also perfectly apply to all use cases of the bank customer.

Furthermore, this main success sequence is completed by the A1 (temporarily incorrect pin number), E1 (invalid card) and E2 (conclusively incorrect pin number) alternative or error sequences.

We can therefore rightfully identify a new use case included in the previous ones that we will call *Authenticate,* and which contains the sequences quoted above. This will allow us to remove all these redundant textual descriptions from the other use cases by concentrating better on their functional specificities.

In UML, this mandatory include relationship between use cases is shown by a dashed arrow with an open arrowhead from the base use case to the included use case. The arrow is labelled with the keyword `<<include>>`, as indicated on the following diagram.

16. From the OMG document: "Unified Modeling Language: Superstructure (version 2.0)".

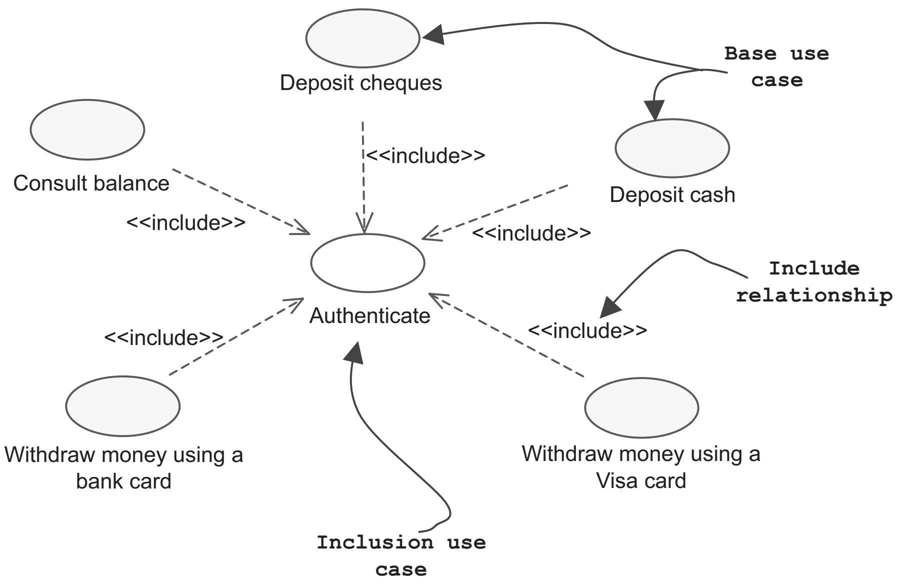

Figure 1.13 Include relationship between use cases

Note that this solution assumes that the ATM users have to re-authenticate themselves for each kind of transaction. If that is not what we require, we should instead envisage the "Authenticate" use case as a precondition for all the others, but not as an included use case.

Let's continue our analysis with the *extend*: a relationship from an extension use case to a base use case, specifying how the behaviour defined for the extension use case augments (subject to conditions specified in the extension) the behaviour defined for the base use case. The behaviour 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 behaviour of the extension use case.[17] 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 behaviour from the mandatory behaviour.

17. From the OMG document: "Unified Modeling Language: Superstructure (version 2.0)".

* * *

1.12 By extrapolating on the initial requirements, identify an extend relationship between two use cases of the bank customer.

Answer 1.12

When re-examining the withdraw money issue, it did not take us long to notice that the bank customer applies almost the same main success sequence as the Visa CardHolder. However, as a customer, he or she also has access to the other use cases: why not allow him or her to consult his or her balance just before he or she selects the desired withdrawal amount? The customer could then change the desired amount according to what is left in his or her account.

If we keep this new functional requirement, all we have to do to model it in UML is add an optional extend relationship, as demonstrated on the following figure.

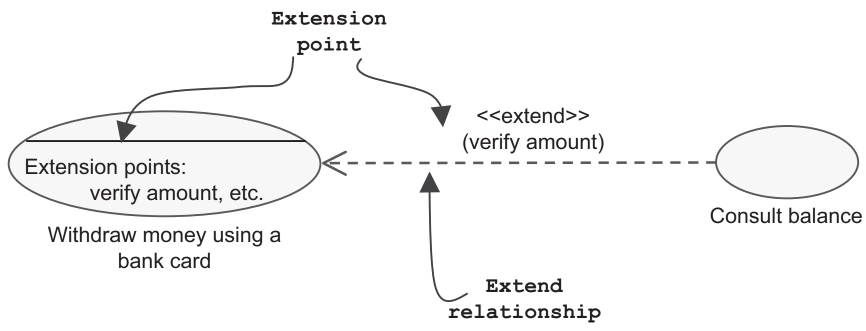

Figure 1.14 Extend relationship between use cases

The two use cases can, of course, be executed independently, but *Consult balance* can also be inserted within *Withdraw money using a bank card*, at the *Verify amount* extension point. This extension point must be declared in the textual description, for example, by modifying the nominal sequence, as we have done here:

...

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.

...

Finally, let's continue with the *generalisation* relationship: the child use cases inherit the behaviour and meaning of their shared parent use case. Nevertheless, each can include additional specific interactions, or modify the interactions that they have inherited. We use this relationship to formalise any important variations on the same use case.

*** 1.13 Identify a generalisation relationship that involves two use cases of the bank customer.

Answer 1.13

Let's consider the following two use cases: *Deposit cash* and *Deposit cheques*.

They both involve the same actors: the *Bank customer* as the primary actor and the *Bank IS* as the secondary actor. But in particular, they say the same thing: the possibility offered to a bank customer to deposit money using the ATM. Whether this transaction entails inserting the notes in a note reader, or simply depositing an envelope containing one or more cheques is not important. The result will be similar, that is to say, a credit line will be entered on the customer's account.

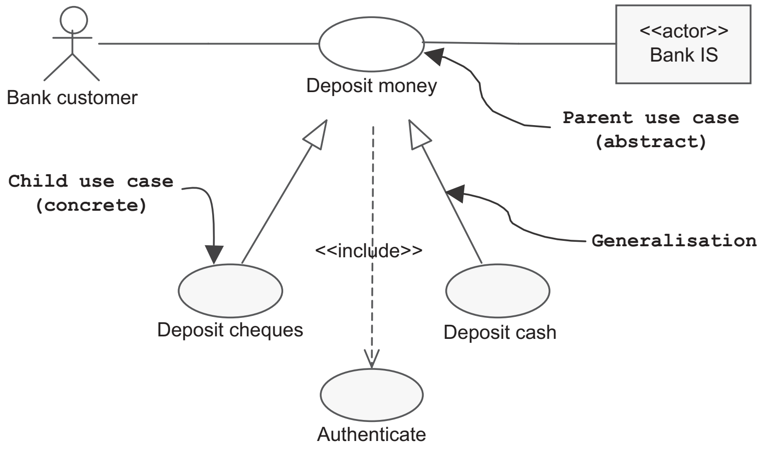

Figure 1.15 Generalisation relationship between use cases

Yet, the details of the sequences will vary considerably: for example, cash deposits require a device that will recognise the various notes, with interactions linked to each time notes are inserted, possible errors (unrecognisable note, etc.) and the end of the transaction. It is also likely that the system for the upkeep of accounts (which belongs to the *Bank IS*) is informed of the deposit in real time in order to credit the account. As for cheque deposits, though, these will involve a bank clerk carrying out a manual verification well after the transaction has finished.

In order to formalise this functional unit, whilst simultaneously retaining the possibility of describing the differences at sequence level, we use the generalisation relationship. All you have to do is add a generalised use case called *Deposit money*. This new case has the special feature of being abstract (which is shown by the *italics*), as it cannot be directly instantiated, but instead, only through one of its two specialised cases.

Notice also that the include relationship with the *Authenticate* use case is now automatically shared by the children use cases.

So, what happens to our use case diagram with all these additions? It is now so complex (compared to Figure 1.4) that it would be deceptive to think that it might be readable in a single page, as the following diagram shows.

To improve our model, we will therefore organise the use cases and reassemble them into coherent groups. To do this, we use the general-purpose mechanism for grouping elements in UML, which is called *package*.

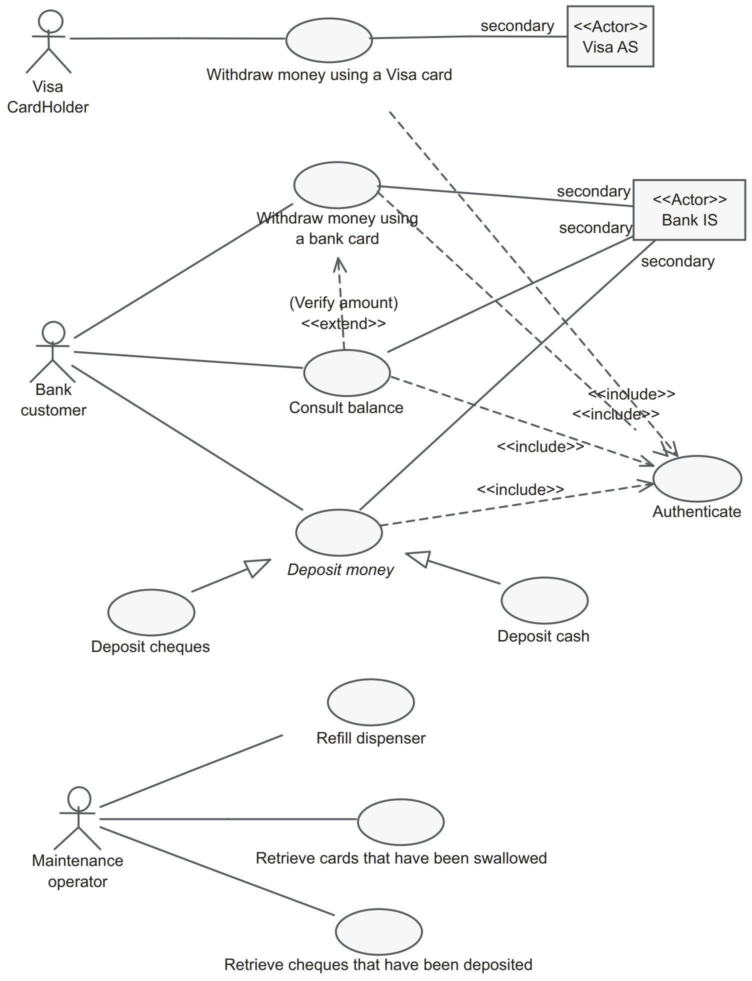

Figure 1.16 Complete use case diagram of the ATM

* *

1.14 Propose structuring the use cases of the ATM into packages. Once you have done that, then develop one use case diagram for each package.

Answer 1.14

There are several possible strategies: proceed with grouping by actor, by functional domain, etc. In our example, grouping use cases by primary actor is natural, as this also allows the secondary actors to be distributed.

The inclusion use case, Authenticate, is placed in a separate package as a shared support service, in order to distinguish it from the real functional cases which include it. The dependency arrows between packages synthesise the relationship between the contained use cases. The following diagram presents the proposed structuring of the use cases by making the primary actor appear in front of each package to remind us which actor is connected to which package.[18] Note that the use of double-headed filled arrows to connect packages to their primary actors is not UML syntax, but here only to explain the packaging.

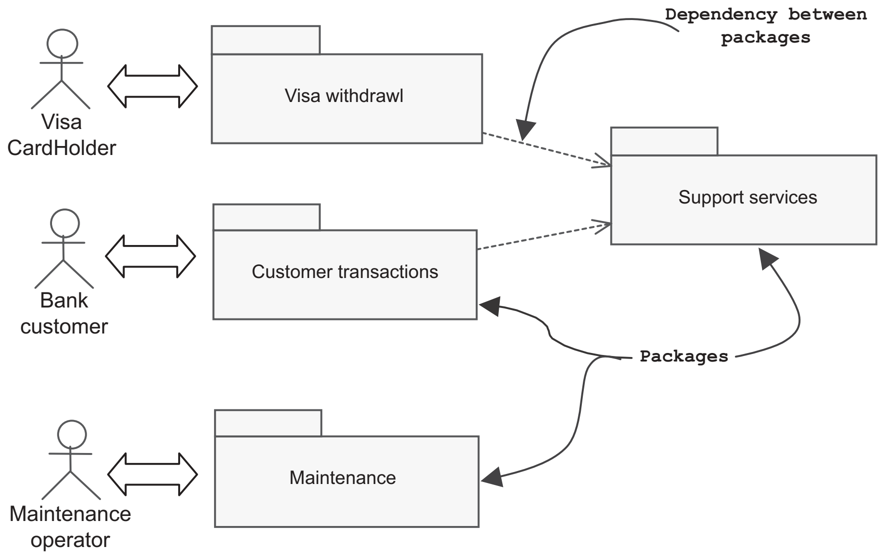

Figure 1.17 Structuring of the ATM use cases

18. UML 2.0 has just added the concept of "package diagram": A diagram that depicts how model elements are organised into packages and the dependencies among them, including package imports and package extensions. Figure 1.17 belongs to this kind of organisational diagram.

We can now create a use case diagram for each of the three main packages.

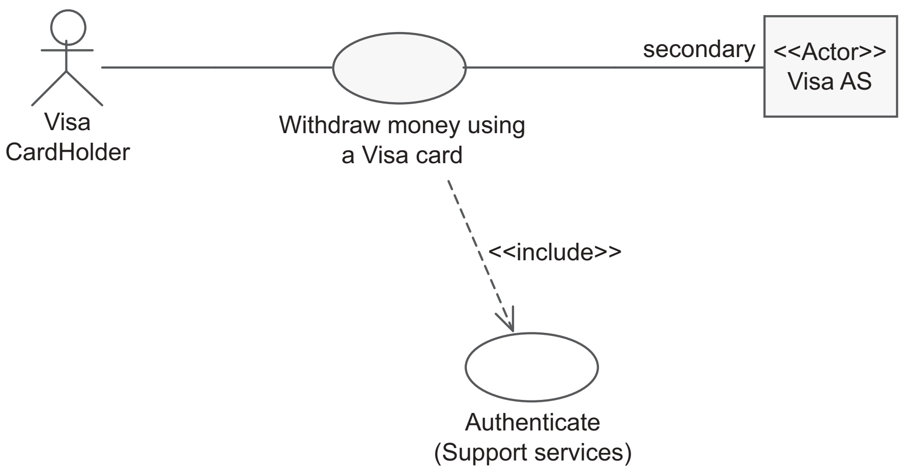
Visa CardHolder
Withdraw money using a Visa card
secondary
<<Actor>> Visa AS
<<include>>
Authenticate (Support services)

Figure 1.18 Use case diagram of the Visa withdrawal package

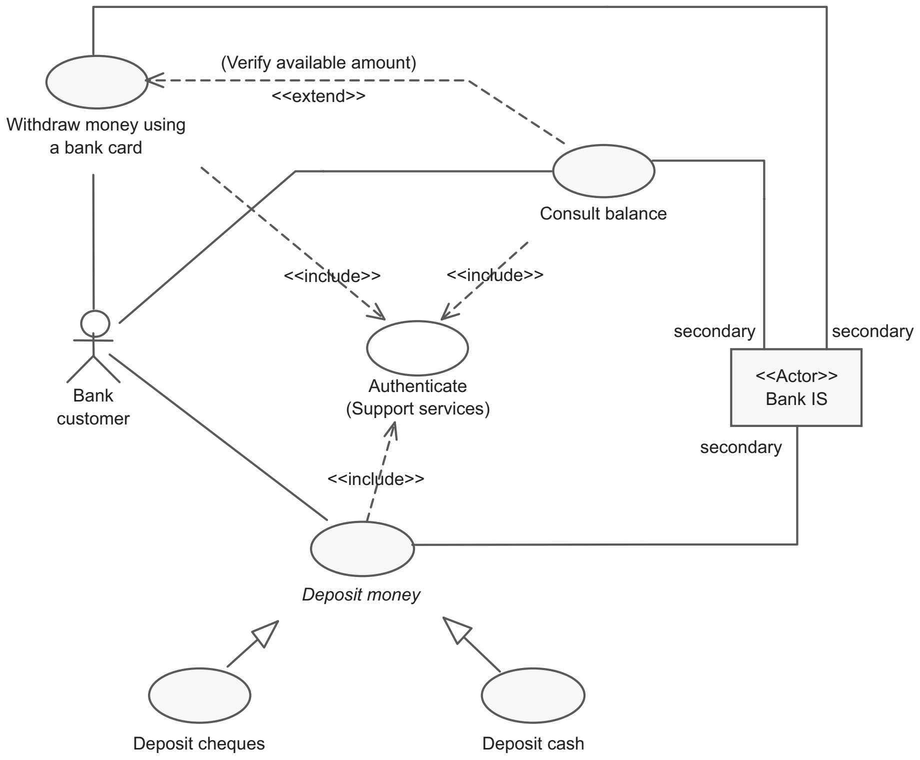

Figure 1.19 Use case diagram of the Customer transactions package

Note that Figure 1.19 is still complex, mainly because we chose to show graphically the relationship between use cases. You must be aware that these UML constructs are potentially dangerous in that the more complex syntax makes the diagram less intuitively obvious to read. They can also lead to modelling errors, that is why many practitioners tend to discourage using them. For instance, Rosenberg[19] points out in his "Top 10 Use Case Modelling Errors": *Spend a month deciding whether to use include or extend*! And Cockburn[20] explicitly warns: "if you spend much time studying and worrying about the graphics and the relations, you are expending energy in the wrong place. Put it instead into writing easy-to-read prose."

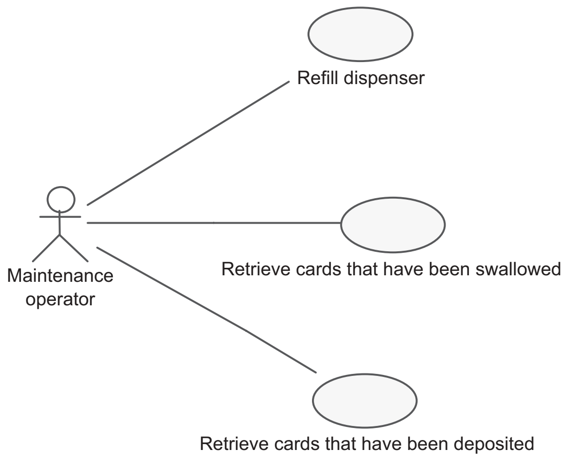
Refill dispenser
Maintenance operator
Retrieve cards that have been swallowed
Retrieve cards that have been deposited

Figure 1.20 Use case diagram of the Maintenance package

Bibliography

[Adolph 02]	*Patterns for Effective Use Cases*, S. Adolph, P. Bramble, Addison-Wesley, 2002.
[Bittner 02]	*Use Case Modeling*, K. Bittner, I. Spence, Addison-Wesley, 2002
[Booch 99]	*The Unified Modeling Language User Guide*, G. Booch, Addison-Wesley, 1999.

19. *Applying Use Case Driven Object Modeling with UML: An Annotated e-Commerce Example*, D. Rosenberg, K. Scott, Addison-Wesley, 2001.

20. *Writing Effective Use Cases*, A. Cockburn, Addison-Wesley, 2001.

[Cockburn 01]	*Writing Effective Use Cases*, A. Cockburn, Addison-Wesley, 2001.
[Fowler 03]	*UML Distilled* (3rd Edition), M. Fowler, K. Scott, Addison Wesley, 2003.
[Jacobson 99]	*The Unified Software Development Process*, I. Jacobson et al., Addison Wesley, 1999.
[Kulak 03]	*Use Cases: Requirements in Context* (2nd Edition), D. Kulak, E. Guiney, Addison-Wesley, 2003.
[Larman 01]	*Applying UML and Patterns, (2nd Edition): An Introduction to Object-Oriented Analysis and Design*, C. Larman, Prentice Hall, 2001.
[Rosenberg 99]	*Use Case Driven Object Modeling with UML*, D. Rosenberg, Addison-Wesley, 1999.
[Rosenberg 01]	*Applying Use Case Driven Object Modeling with UML: An Annotated e-Commerce Example*, D. Rosenberg, K. Scott, Addison-Wesley, 2001.
[Rumbaugh 99]	*The Unified Modeling Language Reference Manual*, J. Rumbaugh, Addison-Wesley, 1999.
[Schneider 01]	*Applying Use Cases: A Practical Guide* (2nd Edition), G. Schneider, J. Winters, Addison-Wesley, 2001.