The «include» and «extend» Relationships in Use Case Models

Introduction

UML defines three stereotypes of association between Use Cases, «include», «extend» and generalisation. For the most part, the popular text books on UML introduce the «include» relationship but give little useful guidance on the «extend» and the generalisation relationships. Confusion arises with project teams as to the correct usage and this document seeks to clarify the situation and make some practical recommendations based on actual project experience.

The first part of the document establishes the basic principles of Use Case modelling as a foundation for exploring the Use Case relationships.

Background

UML is an industry standard for software specification that may be applied to a wide variety of computing systems. The examples in this document are taken from operational business systems and it is left to the reader to decide to what extent the principles described are more or less applicable to other types of problem.

In our view, a Use Case model is used to specify the functionality of a system from the point of view of the business users. Each Use Cases describes a logical task that may be performed by a user of the system. The Use Case description describes the interaction between the system and the outside world. That interaction may be an online transaction where the Actor is a human user. Alternatively a Use Cases may describe the interaction between two systems where no human is involved and in this case the Actor is the external system. Finally the Use Case may describe a piece of background or batch processing, when the Actor is often depicted as a time event; in this case, the dialogue with the outside world tends to be minimal, perhaps just one or two steps. Our examples will focus on online Use Cases with a human Actor, since they are usually more common and contain more complex conditional flows.

The Principles of our Approach

We use UML as an effective tool to create a specification for a system. A development process that is aimed at new build can be adapted quite straightforwardly for enhancing existing systems or evaluating and implementing packages.

For project teams to adopt UML effectively, it is important to be able to derive benefit quickly from UML modelling activities and minimise the burden of learning new techniques. Therefore, in considering the best use of the different types of Use Case associations we apply the Pareto principle (the 80/20 rule) and focus on the basic elements of the notation and defer anything non-essential. This allows us to get as much value as possible from modelling and stay within the orthodox rules of UML.

When modelling operational systems our Use Cases will be defined at a consistent level of granularity. Each primary[1] Use Case corresponds to a logical unit of work, so that as a rule of thumb it is typically,

- performed by one person, in one place, at one time,

[1] For the purposes of this document a primary Use Case is a logical unit of functionality identified as a user requirement of the system. The secondary Use Cases are refinements of the Use Case Model that are identified as included or extending Use Cases.

© Karona Consulting Ltd

- leaves the business data in a consistent state and
- generates some business value.

This standard approach to the granularity of Use Cases gives us more consistency across projects and makes Use Cases a useful metric for estimating project cost. We do not create high-level "summary" or "business" Use Cases; instead we start by modelling the business processes and from there we identify our system Use Cases.

The Structure of a Use Case

A Use Case describes the behaviour of a business system from the business user's point of view[2]. It should describe in plain business terms how the user interacts with the system (assuming it is an online Use Case) and what the system does in response. It does not determine how the system works internally, that is it does not define the implementation.

The description of a Use Case is usually presented in text. There is a good deal of variation in how this is done, especially in the names assigned to the sections that can be included. We will focus here on the description of the Use Case steps, so we will disregard elements such as pre- and post-conditions.

The Use Case steps are usually arranged into a Basic Flow and Alternative Flows.

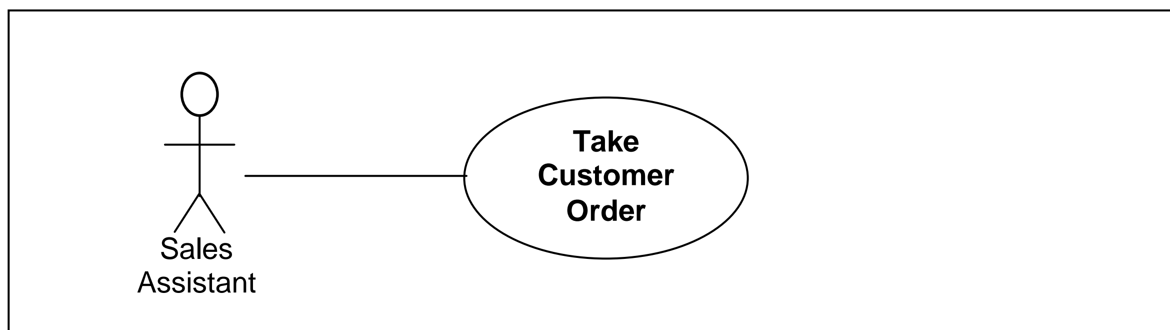
The Basic Flow has other names, such as the *Main Success Scenario* or *Happy Day scenario*. It describes the steps that are required for a straightforward (but not always the simplest) execution of the Use Case. It sets aside the panoply of possibilities within the Use Case so that we can see the essence of the thing. This is helpful to the reader because it allows them to understand quickly what the Use Case is aiming to do.

The Alternative Flows are also known by other names, such as *exceptions* or *extensions*. Each one describes some variation from the steps in the Basic Flow. Alternative Flows can also be overlaid on top of other Alternative Flows. Within a single execution of the Use Case – a single *Scenario* – any number of Alternative Flows may be invoked.

The Use Case steps, both in the Basic and Alternative Flows, are usually numbered. This is not a requirement but it helps the readability of the flows and in particular helps us to define where an Alternative Flow takes effect. Numbering schemes should be kept simple otherwise renumbering becomes increasingly time-consuming as we refine the Use Case.

The UML does not require (or preclude) numbering of Use Case steps. However it specifies that labels may be attached to particular steps so that we can refer to identifiable points in the flow from elsewhere. More on this later.

Example 1 – Simple Use Case



[2] We can use Use Cases in other contexts, for example to describe some common framework code or an infrastructure subsystem. In this case the description of the Use Cases would treat the business application as the Actor and is written from the view point of the application developer, who plays the part of the user. This type of Use Case would never appear in the same diagram as regular business-focussed Use Cases.

© Karona Consulting Ltd

Example 1 cont'd
Use Case: "Take Customer Order"
Basic Flow:

1. Actor enters Customer details
2. Actor enters code for product required
3. System displays Product details
4. Actor enters quantity required
5. Actor enters Payment details
6. System saves Customer Order

Alternative Flows:
[multiple products]
After step 4, when the Actor enters the quantity required,
Repeat steps 2 to 4 for additional Products
Resume at step 5, to enter Payment details

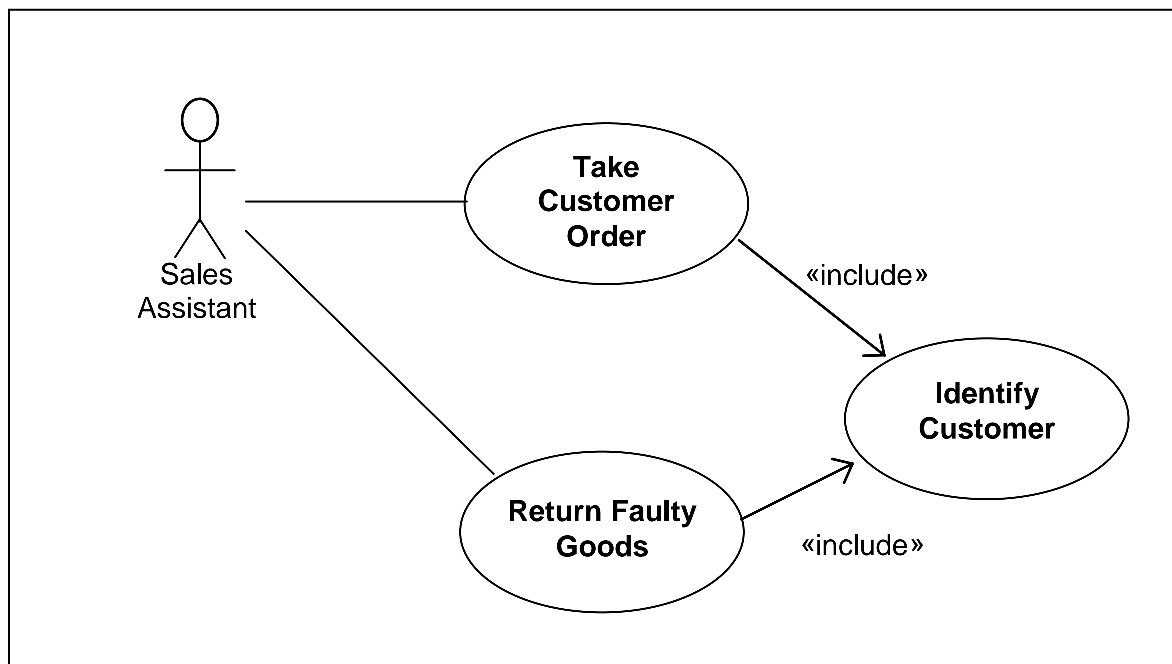
The nature of the «include» relationship

The «include» relationship allows us to include the steps from one Use Case into another. This is valuable when the included steps occur as a recognisable sequence in many different contexts.

In example 1, we need to capture the Customer's details. Following detailed analysis, we may find that there are other Use Cases where we also need to record the Customer's details. Furthermore, will find that there are a number of ways of doing this, for example, we may need to capture them as a new customer or recall them as an existing customer.

So long as the steps required to capture the Customer's details are not trivial, it becomes more economic to extract them into a separate flow, which we may call "Identify Customer".

Example 2 – shared common steps



© Karona Consulting Ltd

Example 2 – cont'd

Use Case: "Identify Customer"

Basic Flow:

1. Actor enters search criteria, surname and postcode
2. System displays matching Customers
3. Actor selects Customer
4. System displays Customer details
5. Actor confirms Customer

Alternative Flows:

[new customer]

After step 2, when the System does not display the required Customer, Actor creates new Customer,

1. Actor selects to add new Customer
2. Actor enters Customer details

Resume at step 5, to confirm Customer

We now modify the base Use Case "Take Customer Order" to refer to the included Use Case "Identify Customer".

Example 2 – cont'd

Use Case: "Take Customer Order"

Basic Flow:

1. Actor records Customer details, **include** (Identify Customer)
2. Actor enters code for Product required
3. System displays Product details
4. Actor enters quantity required
5. Actor enters Payment details
6. System saves Customer Order

Alternative Flows:

[multiple products]

After step 4, when the Actor enters the quantity required,

Repeat steps 2 to 4 for additional Products

Resume at step 5, to enter Payment details

So when we execute the Use Case "Take Customer Order" step 1 tells us to execute all the steps of "Identify Customer", including the possibility of all its Alternative Flows.

The syntax for the **include** statement is not fixed. The name of the included Use Case should be highlighted by placing it in parentheses or quotation marks.

Notice that the base Use Case "Take Customer Order" refers to the included Use Case "Identify Customer" but the included Use Case "Identify Customer" does not refer to the base Use Case "Take Customer Order".

Also note that we can infer who the Actor is for "Identify Customer" from his or her interaction with "Take Customer Order".

© Karona Consulting Ltd

If the included Use Case constitutes a meaningful sequence of steps that can be performed in its own right[3], we might also show the Actor associated with it directly.

The nature of the «extend» relationship

The «extend» relationship allows us to modify the behaviour of the base Use Case. Suppose we want to sell products that are made to order and require a degree of customer specification. For these products we will need to record the customer's additional requirements, such as size and colour. In this case we are adding something extra to the flow of the base Use Case.

We could do this as an Alternative Flow, thus:

Example 3

Use Case: "Take Customer Order"

Basic Flow:

1. Actor records Customer details **include** (Identify Customer)
2. Actor enters code for product required
3. System displays product details
4. Actor enters quantity required
5. Actor enters payment details
6. System saves Customer Order

Alternative Flows:

[multiple products]

After step 4, when the Actor enters the quantity required,

Repeat steps 2 to 4 for additional products

Resume at step 5, to enter payment details

[customer specified product]

At step 3, when the System displays the Product details, if the product requires customer specified features,

1. Actor enters customer specified requirements, such as size and colour.

Resume basic flow at step 4, to enter quantity required,

2. At step 6 where the Customer Order is saved, the additional customer-specific product details must also be saved.

However, the new functionality may open up a whole raft of possibilities and there is a danger that the Alternative Flow spawns further sub flows. The Use Case may become bloated and difficult to manage. To avoid this, the «extend» relationship can be used to pull the Alternative Flow and its sub-flows out into a new Use Case.

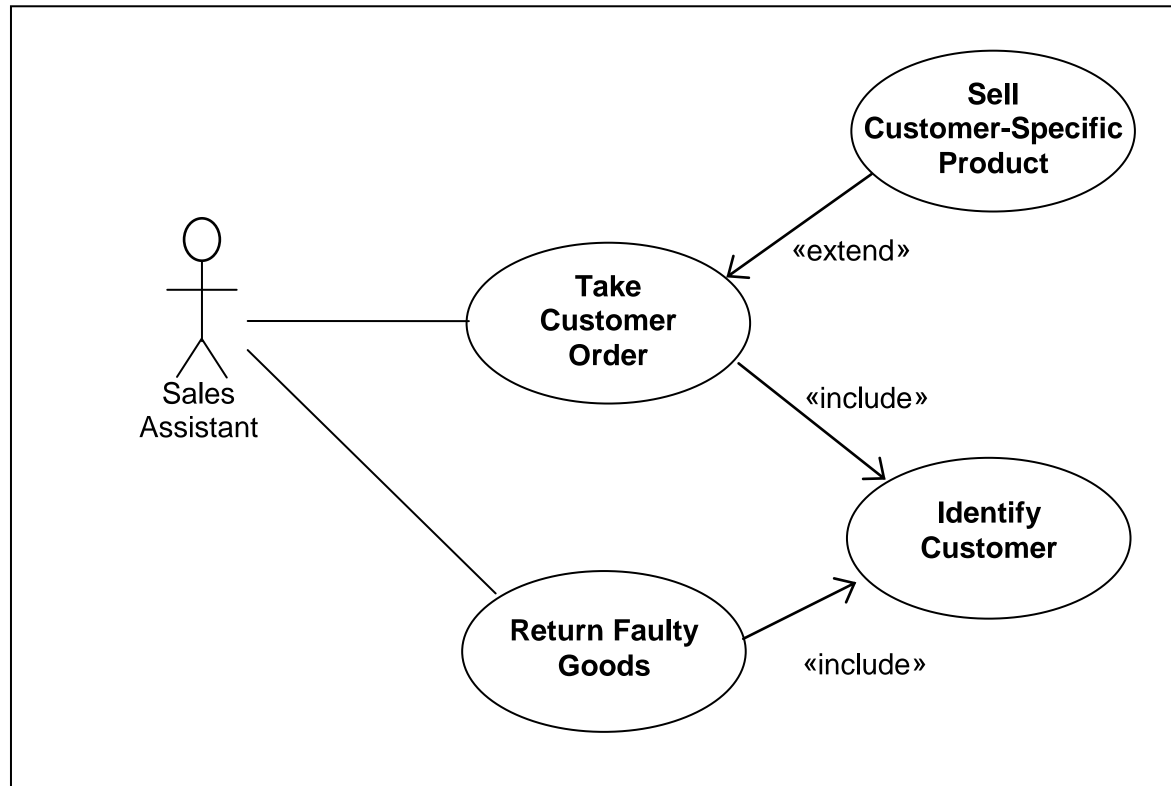
The «extend» relationship says that we execute the primary Use Case but when we get to a specified point in the flow, if the right conditions are met, we perform some different steps. Clearly this is very similar to an Alternative Flow. The advantage is that the Alternative Flow and any dependent sub-flows have been moved into a separate Use Case.

[3] An example of this might be "Find Product" which might occur as part of the Use Cases "Return Faulty Goods" and "Take Customer Order" but could also be used as a free-standing product look-up function.

© Karona Consulting Ltd

Of course this is only advantageous if the flow of the extending Use Case is reasonably self-contained and somewhat divorced from the main flow. If the two are too intimately linked, the flow will be difficult to follow when it is split between two descriptions.

Just as with an Alternative Flow, the extending Use Case may specify a range of steps during which the additional steps may be activated. It may also specify that there is more than one point in the flow where the steps are varied.



Example 4 – extending the primary Use Case

Example 4 – cont'd
Use Case: "Sell Customer-Specific Product" Basic Flow: At step 3 in the basic flow of "Sell Customer-Specific Product", when the System displays the Product details, if the product requires customer specified features, 1. Actor enters customer specified requirements, such as size and colour. Resume "Sell Customer-Specific Product" basic flow at step 4, to enter quantity required, 2. At step 6 where the Customer Order is saved, the additional customer-specific product details must also be saved.

UML tells us that we should create reference points in the flow of the base Use Case called "extension points" to say where the extra steps must be inserted. It also says that these can be overlaid on top of the definition of the primary Use Case without affecting the primary Use Case. This is not a feature that seems to be supported by CASE tools, so in practice we usually use the step numbers as extension points (as we do with Alternative

© Karona Consulting Ltd

Flows). To counter the problem of steps in the base Use Case being renumbered we also paraphrase the step, so that it can be identified again later.

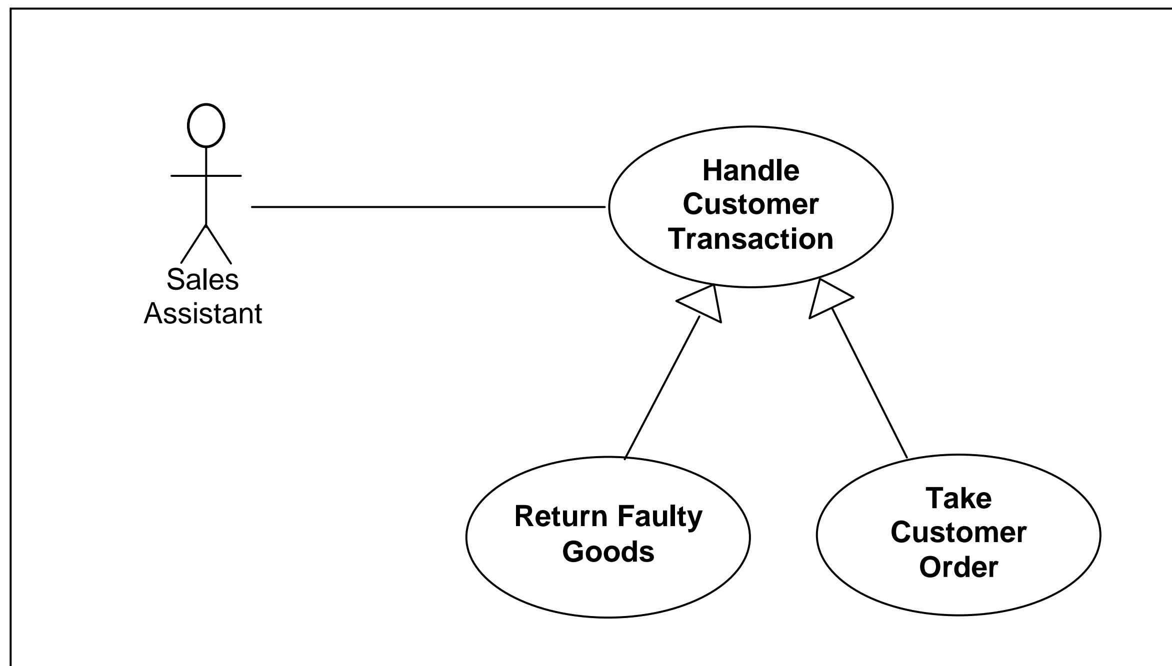
Notice that the extending Use Case refers to (and "knows about") the primary Use Case. The primary Use Case does not have any reference to the extending Use Case. This is clearly stipulated by UML[4].

The nature of the generalisation relationship

A generalisation relationship between Use Cases indicates that the child Use Cases inherit the properties of the parent Use Case.

Examples where this is a valuable aid to creating a useful Use Case model seem to be hard to find. We suspect that the generalisation relationship arises more from a theoretical consideration than any practical need.

The following example is provided only for completeness.



In this example, we would be indicating that there are some common steps for all Use Cases that handle customer transactions and that the child Use Cases "Return Faulty Goods" and "Take Customer Order" have additional steps that fit into or around them.

In order to specify the additional steps for the child Use Cases you need to have extension points declared against the parent Use Case.

When to use the «include» relationship

The time to use the «include» relationship is *after* you have completed at least the first cut basic flows for all your main Use Cases. You can now look at the Use Cases and identify common sequences of user-system interaction. Do not attempt to do this earlier as it will be hard to justify the included Use Cases and they are likely to be more of a distraction than a help at this stage.

Each included Use Case is an essential part of the complete description of the primary Use Case.

---

[4] We could mark the extension points in the base Use Case to show where it is extended, for example by appending some sort of tag to a step like, "[extended by "Sell Customer-Specific Product"]". This breaks the rule that a base Use Case has no reference to the extending Use Case. In practice, however, we can see no disadvantage in doing it this way.

If you have overlooked some common user interaction during the initial Use Case analysis, don't worry – it is likely to jump out at you when you start to consider the user interface. In any case, a missed opportunity for an included Use Case has little more significance than some duplicated effort for the analyst and is unlikely to harm the system design.

Remember when applying «include» relationships that the aim is to simplify the model not to complicate it.

When to use the «extend» relationship

As with the «include» relationship, the time to use the «extend» relationship is *after* you have completed the first cut description of all your main Use Cases. Do not attempt to do it before this as the validity of the extending Use Cases will be questionable.

In fact you should be able to produce a pretty complete first cut Use Case Model without any «extend» relationships and we would advise you to do this. The extending Use Cases represent additional functionality, *without which the model is consistent in itself*.

The «extend» relationship should be applied to a well-formed first cut Use Case model in the following circumstances:

1. A Use Case is too big. If a Use Case description, with all its alternative flows is becoming unmanageable, use this as a strategy to break it up. Take a substantial alternative path, that is reasonably self-contained and make it an extending Use Case.

   For example, the Use Case "Sell Basket of Goods" in a Point Of Sale system is notoriously complex. To simplify the main Use Case we can pull out "Give Cashback" as an extending Use Case. Note also in this example, that "Register Item" and "Take Payment" will have probably dropped out as included Use Cases.

2. The primary Use Case is something that for one reason or another we don't want to change. It may be that the primary Use Case is part of an existing system implementation and perhaps there is no existing Use Case model for that system but we want to create a Use Case description for the new functionality. Using an «extend» relationship helps us to separate out our specification of the new functionality from the old.

   If it doesn't already exist, create a simple Use Case description for the basic flow of the existing Use Case so that we can see where the new Use Case occurs and leave it at that. Now document the new Use Case fully.

   An example of this was the "Sell Customer Specific Goods" in example 4. The Customer Specific Goods in question were made-to-order window blinds. This was new functionality being added to an existing system which required substantial new business logic. The existing flow of the main Use Case was extended in one place for capturing and validating the customer's requirements and another for saving the additional order data.

3. Where we want to delay delivery of some flows within the Use Case. We normally use Use Cases to define the scope of our increments and occasionally this causes us a problem if we want to partially implement a function and withhold some features until a later increment. What we do is pull out the delayed alternative flows into extending Use Cases so that we can track their delivery independently of the main Use Case.

   This is more likely to occur in the first increment where we are trying to create a small increment of realistic business functionality but the core Use Cases have a lot of alternative flows.

© Karona Consulting Ltd

Bear in mind that the «extend» relationship is more likely to cause confusion and disagreement than almost any other area of your UML analysis model. These debates can be time-consuming, so approach it with caution and use it sparingly.

When to use the generalisation relationship

Never. We haven't found a case yet which justifies the existence of this notation. We will be happy to receive an example if anyone has one to offer.

When «include» and «extend» meet

There are some cases where a strong argument can be made either way for using an «include» or an «extend» relationship. Characteristically these occur where the additional steps are optional.

The following touchstone questions are offered for guidance but should be applied judiciously:

1. Do the additional steps constitute a single contiguous set of steps?

Yes, you are adding steps at one place in the flow – use «include».

No, you need to modify the primary Use Case in more than one place – use «extend».

2. Do the additional steps make sense on their own?

Yes – use «include».

No – use «extend».

In an included Use Case the additional steps may or may not be performable in isolation but they are likely to have a degree of coherence on their own. The steps of the extending Use Case are more likely to be an incomplete fragment or sequence of fragments.

3. Do the additional steps occur in more than one primary Use Case?

Yes – use «include».

No – use «extend».

Although UML clearly allows that an extending Use Case *may* extend more than one primary Use Case, it is rarely likely to be used[5]. To achieve it, each base Use Case would have to support the same extension point or points referred to by the extending Use Case. All the primary Use Cases must have a logically similar step (or steps) which needs to be extended. Taken with a "yes" answer to Question 1 (the extension applies to only one extension point) you could achieve the same result with an «include» relationship and this might be easier to manage without breaking the rules about the direction of the reference.

Summary

1. Do not try to identify «include» and «extend» relationships in your first cut model.
2. Use «include» to abstract sequences of steps out of the primary Use Cases that are useful to the analyst to avoid repetitious descriptions.
3. Use «extend», sparingly, to overlay additional functionality on top of a well-formed Use Case to divide up the complexity of a large and complex Use Case.

[5] An «extend» Use Case may however extend a base Use Case and a Use Case that the base includes. In this case the two extended Use Cases are both part of the same sequence of user actions, so it is quite reasonable to want to extend in both parts of the sequence. This is however a fairly complicated situation and unlikely to occur in most Use Case models.

© Karona Consulting Ltd