Expansion of Existing Use Cases Using Extend Relationship

Prashant, Nidhi Sharma, Achint Gupta

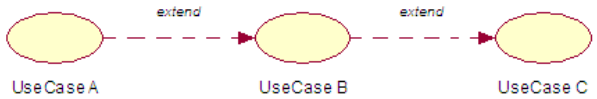
Gurgaon College of Engg., Gurgaon

Abstract:
In this paper, we have discussed the Extend relationship for the use cases, when subsequent modifications are performed on a system or some later versions are added to the existing use cases. The extend relationship makes it easier to read and understand the model. Use case may be extended by more than one use case.

1. INTRODUCTION

A. Extend Relationship

When subsequent iterations in a project are performed or when later versions of a system's use-case model are developed, completely new use cases are often added to the model and new actions are inserted into existing use cases. This means that the existing services are extended with some additional features that did not exist in the previous versions of the model. However, the different stakeholders often do not want to modify the existing use cases because these have already been reviewed and approved. They are only willing to have certain features added to the existing use cases. Therefore, the developers will have to express these additions without modifying the existing use cases. To solve the situation, it is possible to use the extend relationship. An extend relationship states that the flow of a use case is extended by the flow defined in another use case. The original use case is often called the base use case, and the use case capturing the addition is called the extension use case as shown in Figure. 1 . We can compare these labels to those involved in an include relationship in the sense that this labeling is relative to the extend relationship, and not to the use cases as such.



B is a base usecase to A
B is a extension usecase to C

Figure 1. The terms base use case and extension use case are relative terms

The fact that the relationship is defined from the extension use case to the base use case that is, from the new use case to the already existing one makes it very useful. The direction implies that the extension use case is dependent on the base use case, whereas the base use case is in fact independent of the extension. In other words, the dependency goes to the base use case that is, in the opposite direction compared with the include situation where the dependency goes from the base use case. Therefore, an extend relationship can be added to a model without affecting the base use case; that is, the definition of the base use case will not be modified at all when the sub flow of the extension use case is added. We have considered an example of a warehouse (fig.2.). In the business where this system is used, there are, among others, managers and salespersons. We have two business roles: Manager and Salesperson.

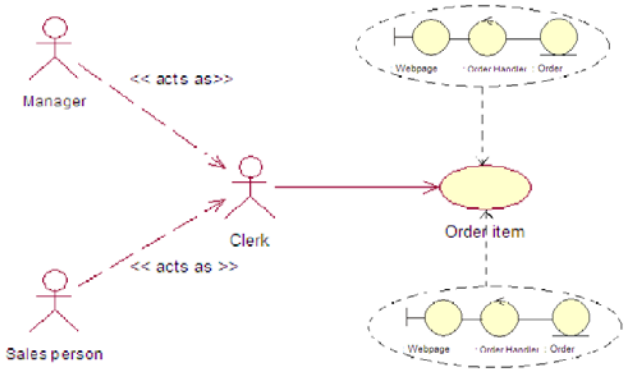


Fig. 2. Example of a Warehouse.

In the example, both managers and salespersons expedite orders that customers want to place. Therefore, from the business roles' perspective, both of them are performing the Order Item use case. There is only one actor, called Clerk, interacting with the Order Item use case as shown in Fig. 3.
The use case receives the order information from the Clerk, creates the order, and prints it to the Clerk. In the warehouse system example, The use case Order Item that describes how a collection of items is ordered, how the registered numbers of these items are decreased in the system, and how a pick list is generated and sent to the warehouse personnel instructing them to deliver the items to the customer. In the first version of the system, the head buyer of the warehouse must manually check whether the number of an item is running low; if so, it must be

Manuscript received September 5, 2013
Manuscript revised September 20, 2013

restocked. Obviously, it would be desirable that the system automatically informs the head buyer when an item has to be restocked. Therefore, in the next version of the system, a new use case, Restock Item, is added to the use-case model. The added functionality implies that when the registered number of an item falls below a specified threshold, a restock order is generated and sent to the warehouse's head buyer. The Restock Item use case has an extend relationship to the Order Item use case stating that when the latter use case is performed, the actions described in Restock Item will be inserted into the performed sequence of actions (Fig. 4.)

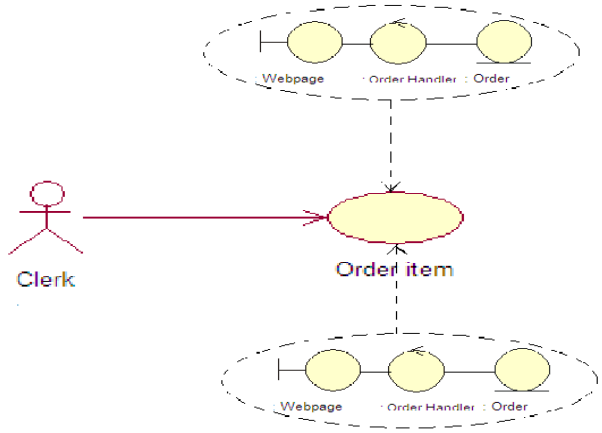


Fig. 3. Use case of Warehouse

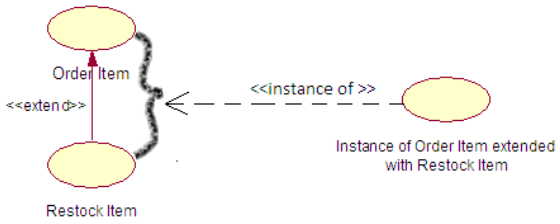


Fig. 4. The use-case instance to the right follows the description of the Order Item use case extended by the description of the Restock Item use case, as indicated by the hatching.

In the first version of the warehouse system, the use-case instance will perform the actions described in the Order Item use case, whereas in the second, enhanced version, the use-case instance will perform the actions of Order Item extended by the actions of Restock Item. There will not be two communicating use-case instances, because use-case instances do not communicate with each other, there is only one flow of events; in the latter version, this flow will include the actions modeled by two use cases. The extend relationship is also used when the flow of a use case includes a part that from a conceptual point of view does not belong to the rest of the flow in other words, when the flow includes a part that, in some sense, is orthogonal to the rest of the flow. This could be logging, for example, which in itself usually has nothing to do with what is being logged; in this sense, it is orthogonal to the rest of the flow. How logging is done can be described separately (Fig.5.)

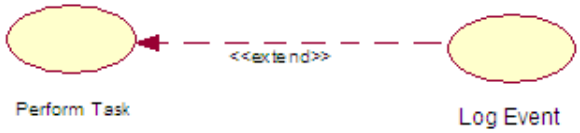


Fig. 5. Modeling behavior that conceptually does not belong together with the rest of the behavior of a use case is often done using an extension use case.

Used in this way, the extend relationship makes it easier to read and understand the model. In our telephone exchange system, assume that all telephone calls are to be charged. Hence, the flow of Local Call must include the actions that cause the charging to take place. However, the information about the charging procedure may be distracting to readers of the Local Call use case. Instead, we describe charging in a separate use case extending Local Call (Fig.6.).

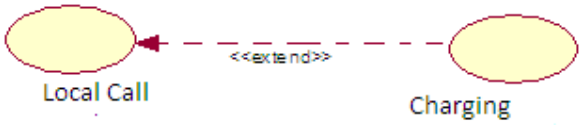


Fig. 6. Extracting behavior into an extension use case may increase readability and understandability.

One important motivation for the extend relationship is that it makes it possible to model services that are optional in a configuration of (an installation of) the system, enabling the customers buying the system to decide whether to include a certain service. In the warehouse system example, there can be one basic version of the system and a collection of additional services, including automatic restocking. Different warehouses can buy different configurations of the system by selecting different sets of use cases (Fig.7.). This implies that the inclusion or exclusion of optional services must not affect the other parts of the model. Because we have related Restock Item to Order Item using an extend relationship, we can add as well as remove it without influencing the rest of the model, thus obtaining the desired structure of the model.

If for some reason the use-case model must be modified depending on whether some optional parts are supplied, the model will of course have to be structured differently.

One use case may, of course, extend several other use cases like the Charging use case, which probably extends all the use cases that model the performance of some kind of telephone call, as is shown in Fig. 8. There will be an extend relationship from the Charging use case to each of the extended use cases, which means that each of them is extended with the sequence of actions described by the Charging use case.

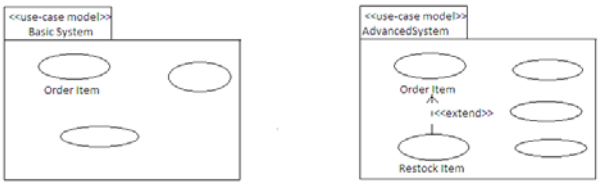


Fig. 7. In different configurations of the system, optional use cases can be added. Some of these may expand mandatory use cases using extend relationships.

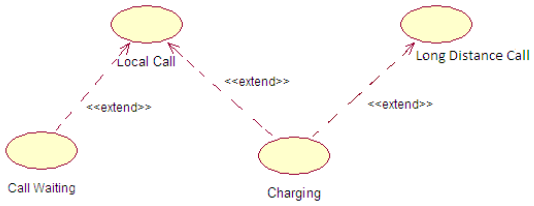


Fig. 8. One use case may extend several use cases, and several use cases may extend the same (base) use case.

Furthermore, one use case may be extended by more than one use case. The Local Call use case may be extended not only by the Charging use case but also by, for example, the Call Waiting, the Present Caller Line Identification, and the Survey Quality of Call use cases.

B. Characteristics of the Extend Relationship

As mentioned previously, the relationship is directed from the extension use case to the base use case. This implies that the base use case is independent of the extension. Only the extending use case is dependent on the extend relationship. This means that the extend relationship can be used only to model additional behavior it cannot be used for modifying or removing behavior within the base use case. The rationale for this restriction is to make sure that the base use case will not have to be reviewed and approved after the introduction of an extension. When we extract behavior from a base use case and put it in an extension use case, the base use case must still be complete, understandable, and meaningful in itself. There must not be an apparent hole in the sequence of the base use case; that is, it must not be possible to conclude by just looking at the description of the base use case that there has to be an extension of this use case, because something is apparently missing. Remember that the base use case is independent of any extensions and hence, at least in theory, it must be possible to perform just the base use case without any extension. If the base use case is of no value to the stakeholders without the insertion of an extension, it is not a justifiable use case and the model should be restructured. Similarly, if the description of the base use case necessitates a reference to an extending use case, it is evident that the extend relationship is inappropriate. Again, the base use case must be independent of the existence of extension use cases. In this situation, two straightforward solutions exist. One is to merge the behavior of the extension use case with the behavior of the base use case. In cases where the extension use case is shared with other base use cases, the solution is to use an include relationship instead of the extend relationship. This is appropriate because in this case the additional parts are referenced from the base use case.

C. Extension Points

Clearly, it is not enough to state only that the behavior of the extension use case is to be added into the sequence of the base use case. The exact location in the sequence of actions where the extending sequence is to be inserted must also be defined. A straightforward way to do this is to reference an explicit location in the base use case's sequence. This is how this was done in the early days of use-case modeling. It soon became obvious that an indirect reference would be preferable, however. There were two reasons for this change. One was that to understand where to insert the extending behavior, the use-case modeler must read and understand the detailed description of the base use case to find exactly where the extension was to take place, which is not always an easy task. The other reason was that if the base use case were later to be modified, the explicit reference might prove to have become invalid: The referenced location might have been removed or might no longer be the desired location. Therefore, extension points were introduced. An extension point declared in a use case consists of a name and a reference to a location in the sequence of actions of the use case where it may be extended (Fig. 9). Therefore, the extension point belongs to the base use case, implying that the reference to the exact location in the flow is encapsulated within that use case.

An extend relationship will now not only identify the extending and the base use cases, but it will also define at which extension point in the base use case the behavior of the extension use case is to be inserted. When an extend relationship is to be added, use-case modelers check the list of extension points declared in the base use case to identify at which point the additional behavior is to be inserted. In this way, they have to read and understand

only the extension points, not the complete description of the base use case. Therefore, the use case might be slightly reorganized without affecting the extending use cases. A major reorganization of a use case may affect all the use cases that have relationships to it. The name of an extension point should describe what happens at this location in the use case, not the actual location in the sequence of actions. In this way, the extension points will be easier to understand and the behavior of the use case will be kept encapsulated. The name of the extension point must not reveal what is to be inserted at that point, because that would make the extended use case dependent on the extending use case.

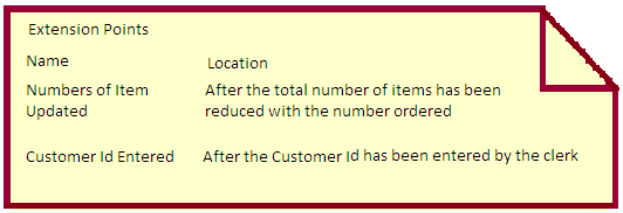


Fig. 9. Two of the extension points defined in the Order Item use case. Each of them has a name and references a location in the flow of the use case.

In Fig.9 the extension points of the use case Order Item are enumerated. Assume that this use case is extended with the Restock Item use case to check whether the number of items is below a given threshold so that additional items must be purchased. The Restock Item use case extends the Order Item use case at the point where the total number of the relevant items is updated. A common mistake is to name the extension point Check for Restocking, because it indicates what is to be inserted. The proper name states what happens in the base use case, which in this case is Number of Items Updated. Another advantage of extension points, often forgotten, is that they also make it possible in the implementation to prepare for extensions that might occur in the future.

D. Conditional Extensions

The extend relationship has another useful property: a condition is connected to it. This condition determines whether the extension is to be inserted when performance of the use case's flow reaches the extension point referenced by the extend relationship. If the condition then evaluates to true, the extending flow is inserted into the flow of the base use case, and the result is a flow complying with the description of the base use case and the description of the extending use case. Again, there is no communication, no calling of other use cases, no delegation, no invoking of other use cases. The condition can be stated in terms of both values inside the system and events caused by actors of the system. An example of the former is the extend relationship from the Restock Item use case in the warehouse model, where the generation of the restocking order takes place only if the number of the specific item is lower than the predefined threshold level (see the left part of Figure 8.8). This indicates that the Restock Item extension will not be inserted if the current number of the item exceeds the threshold level.

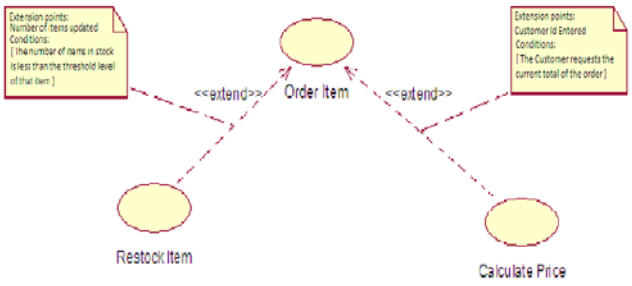


Fig.10. An extend relationship has a condition that states under what circumstances the extension is to take place. The condition can include references to information inside the system and to interactions between an actor and the system.

The warehouse model also contains an example of an extend relationship where the condition is stated in terms of actor interaction. If the customer, when the Order Item use case is performed, wants to know the total cost of the items currently ordered, the customer can request that the system calculate the price, for example, by clicking a button. By defining an extend relationship from the Calculate Price use case to the Order Item use case, and by defining the condition to be The Customer requests the current total of the order, we get a model that captures this situation (Fig. 10.). If there is a use-case instance performing the sequence of actions described by the Order Item use case, and the customer presses the button that is, requests that the sum total be stated the condition of the extend relationship is fulfilled, which in turn means that the sequence of Calculate Price is inserted into the use-case instance. The conditions of extend relationships are usually not shown in diagrams, simply because they make the diagrams messy and harder to read, but they may be included if that adds clarity to the diagram. The fact that an extend relationship is conditional has led to the misunderstanding that an extend relationship is appropriate in any situation where the performance of a certain subflow is ruled by a condition. However, a condition is not always as useful an indicator as one might think. Instead, the criterion for determining whether an extend relationship can be used is whether the part proposed to be factored out can be extracted without making the remaining behavior incomplete and therefore not properly expressed as a use case. It turns out that there is often a condition involved where part of the use case can be extracted into an extension use case. For the convenience of the developer, the definition of the extend relationship therefore includes a condition. However,

some extensions' conditions are always true. In the telephone exchange example, Charging is always inserted into Local Call; therefore, the condition of this extend relationship is true. We sometimes see models where the condition in such situations is left out. As always with implicit information like this, the risk for misunderstanding is obvious. We therefore strongly recommend that use-case modelers explicitly state also those cases where the condition is true.

E. Documentation Of The Extend Relationship

An extend relationship is documented in the description of the extending use case. Note that this is the only place available, because the base use case must include no information about the extension. The relationship should be documented as part of the flow description, explaining how the insertion is initiated (Fig. 11.). If the use case is extending several use cases, each extend relationship should be given a paragraph of its own. It should consist of a reference to the use case that is to be extended, a reference to the extension point where the additional behavior is to be inserted, and the condition that must be fulfilled if the extension is to take place.

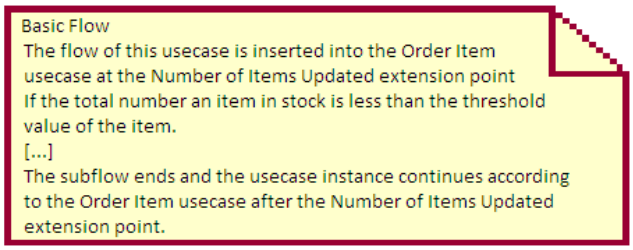


Fig. 11. The description of Restock Item includes a description of the extend relationship as a description of where and why the flow is added. The end of the flow description is also slightly affected by the extend relationship.

Likewise, for each base use case the extension use case extends, there is a paragraph at the end of the description of the flow. (Fig. 11.) Apart from the start and the end of the flow, an extension use case is described as an ordinary use case. Note, however, that an extension use case usually models only part of a usage and not a complete usage, implying that an extension use case is often abstract. Just as for inclusion use cases, this means that the flow of an abstract extension use case does not necessarily start with an input from an actor; nor is it required to include any output or to have a well-defined end of the sequence of actions. The base use case is described as an ordinary use case, and it must not include anything that depends on the fact that there are extension use cases to be inserted into it. The extension points are defined entirely within the base use case, and they are therefore documented in the description of that use case (Fig. 9.). This may seem to contradict the fact that the base use case should be independent of any extensions. However, an extension point does not reveal whether something (and if so, what) may be inserted at that extension point. It is therefore very important to assign to extension points names that are completely understandable within the use case itself, and that in a simple way describe their location in the use-case flow.

2. DEPENDENCIES BETWEEN EXTENDING USE CASES

As stated previously, the base use case is independent of the extension use cases, and the extension use cases and their extend relationships are defined independently of each other. If more than one extend relationship refers to the same extension point in a use case, it is therefore not possible to deduce in what order the extensions will take place if at all, depending on the evaluation of their conditions. Consequently, extend relationships can reference the same extension point only if no particular order is required. Otherwise, different extension points must be used, so that the order in which the extension use cases are to be inserted into the base use case is defined by the locations referenced by the extension points (Fig. 12.).

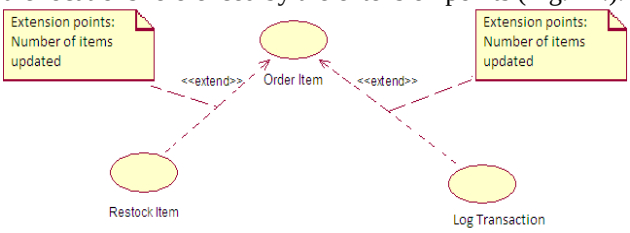


Fig. 12. The order in which Restock Item and Log Transaction will be inserted into Order Item is not deterministic, because they both reference the same extension point in this model. If the transaction must be logged before the item is restocked, Log Transaction must be inserted at an extension point located at a place defined earlier in the flow than the extension point Number of Items Updated.

3. CONCLUSION

Here in this paper, we have studied and explained the extend relation for a use-case patterns that is proven useful when developing maintainable and reusable use-case models. These patterns focus on designs and techniques used in high-quality models. Further, The extend relationship makes it easier to read and understand the model. Use case may be extended by more than one use case.

REFERENCES:

[1] Adolph, S., and P. Bramble . 2002. Patterns for effective use cases.Addison-Wesley.

[2] Alexander, C., S. Ishikawa, and M. Silverstein . 1977. A pattern language: towns, buildings, construction. Oxford University Press.
[3] Bass, L., P. Clements, and R. Kazman . 2003. Software architecture in practice. Addison-Wesley.
[4] Bittner, K., and I. Spence . 2002. Use case modeling. Addison-Wesley.
[5] Buschmann, F., R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal . 1996. Pattern-oriented software architecture, volume 1: a system of patterns. John Wiley and Sons.
[6] Jacobson, I. Concepts for modeling large real time systems. Ph.D. thesis, Royal Institute of Technology, Stockholm, Sweden.
[7] Jacobson, I."Object-oriented development in an industrial environment." Proceedings of OOPSLA'87. Sigplan Notices 22(12) :183191.
[8] Jacobson, I. 2003 (March). "Use cases yesterday, today, and tomorrow." The Rational Edge.
[9] Jacobson, I., G. Booch, and J. Rumbaugh . 1999. The unified software development process. Addison-Wesley.
[10] Jacobson, I., M. Christerson, P. Jonsson, and G. Övergaard . 1993. Object-oriented software engineering: a use- case driven approach. Addison-Wesley.

Mrs. Nidhi Sharma is working as Deputy Registrar in Gurgaon College of Engineering., Gurgaon. She did her B.Sc., M.Sc. in Computer Science, MBA, & currently pursuing her Ph.D. in Computer Science. Her area of interest includes Software engineering, MIS, ERP & E-Commerce

Mr. Prashant is working as an Assistant Professor in the Dept. of C.S.E. & I.T. at Gurgaon College of Engineering., Gurgaon. He did his B.tech., M.tech. in IT & currently pursuing his Ph.D. in Computer Engineering. His area of interest includes Software Engineering, Software testing, OOSE.

Mr. Achint Gupta is working as an Assistant Professor in the Dept. of C.S.E. & I.T. at Gurgaon College of Engineering., Gurgaon. He did his B.tech. & M.tech. in IT.