Use case research

http://stackoverflow.com/questions/1696927/whats-is-the-difference-between-include-and-extend-in-use-case-diagram

Relationships are dependencies

The key to Include and extend use case relationships is to realise that, common with the rest of UML, the dotted arrow between use cases is a dependency relationship. I’ll use the terms ‘base’, ‘included’ and ‘extending’ to refer to the use case roles.

include

A base use case is dependent on the included use case(s); without it/them the base use case is incomplete as the included use case(s) represent sub-sequences of the interaction that may happen always OR sometimes. (This is contrary to popular misconception about this, what your use case suggests always happens in the main scenario and sometimes happens in alternate flows simply depends on what you choose as your main scenario; use cases can easily be restructured to represent a different flow as the main scenario and this should not matter).

In the best practice of one way dependency the base use case knows about (and refers to) the included use case, but the included use case shouldn’t ‘know’ about the base use case. This is why included use cases can be: a) base use cases in their own right and b) shared by a number of base use cases.

extend

The extending use case is dependent on the base use case; it literally extends the behaviour described by the base use case. The base use case should be a fully functional use case in its own right (‘include’s included of course) without the extending use case’s additional functionality.

Extending use cases can be used in several situations:

1. The base use case represents the “must have” functionality of a project while the extending use case represents optional (should/could/want) behaviour. This is where the term optional is relevant – optional whether to build/deliver rather than optional whether it sometimes runs as part of the base use case sequence.
2. In phase 1 you can deliver the base use case which meets the requirements at that point, and phase 2 will add additional functionality described by the extending use case. This can contain sequences that are always or sometimes performed after phase 2 is delivered (again contrary to popular misconception).
3. It can be used to extract out sub sequences of the base use case, especially when they represent ‘exceptional’ complex behaviour with its own alternative flows.

One important aspect to consider is that the extending use case can ‘insert’ behaviour in several places in the base use case’s flow, not just in a single place as an included use case does. For this reason it is highly unlikely that an extending use case will be suitable to extend more than one base use case.

As to dependency, the extending use case is dependant on the base use case and is again a one way dependency, i.e. the base use case doesn’t need any reference to the extending use case in the sequence. That doesn’t mean you can’t demonstrate the extension points or add a x-ref to the extending use case elsewhere in the template; but the base use case must be able towork without the extending use case.

SUMMARY

I hope I’ve shown that the common misconception of “includes are always, extends are sometimes” is either wrong or at best simplistic. This version actually makes more sense if you consider all the issues about directionality of the arrows the misconception presents – in the correct model it’s just dependency and doesn’t potentially change if you refactor the use case contents.