Template Use Case Pattern

Hüseyin Angay

Karabash Ltd.

Member of the Appropriate Process Movement

http://www.aptprocess.com

Abstract

There are many cases where we would like to model significantly different sets of interactions that still have underlying similarities. Modelling this functionality in use cases may lead to many nearly identical use cases, which will eventually become an expensive exercise in redundancy. For projects where modelling these requirements as use cases is still necessary, we propose the Template Use Case pattern that will preserve the consistency and benefits of a use case model without the associated overheads for creating and maintaining virtually identical sets of use cases.

Copyright Note

Contents

Introduction

Managing requirements with use cases has become quite commonplace in the last few years. This paper will not discuss their merits or their problems; we will assume that you chose to apply use cases to your project and that you would like to make the most of them.

One area where use cases can become a burden is in documenting a similar series of interactions on different sets of data. The data is different enough to necessitate slightly different interactions, but the resulting use cases are similar enough to benefit from some factoring out of common parts. Keeping these as separate and independent use cases often is not a good solution because the underlying similarity generally implies a commonality in business processes. When the requirements change for one of the use cases, it is very likely that the underlying pattern will also change for all the other similar use cases, leading to expensive model maintenance. That underlying pattern is often an invariant of the business, so it would be useful to capture it as such.

One common solution to this is to keep a separate use case for each distinct interaction, but to factor out the common behaviour in included use cases. This has two disadvantages:

1. We may need to create many small included use cases in order to accommodate all the common parts.
2. The many associations amongst the use cases soon lead to a tangled mess.

We propose an alternative solution where we reduce the number of use cases to the number of essential interactions, that is, the invariants, and we document the variants in a separate but related artefact.

Applying the Template Use Case pattern

Repetition in use cases

It is relatively easy to get caught up in the frenzy to document as many use cases as possible at once and to lose track of the common behaviour between use cases. That is why any team seriously considering to model systems with use cases should set aside enough time to look at commonalities and factor them out to express the invariants of the business and to reduce the maintenance effort in the future. The Functional Architect plays the leading role in this task as he is most likely to have an overall view of the use case model.

There are several indications that use cases suffer from repetition:

- They have similar names.
- They belong to similarly named packages.
- They include or extend the same or similarly named use cases.
- When requirements change, they regularly change together.

The first three indicators may well indicate commonality, but without the last indicator, they may well be signs of a well structured model. The last indicator, though, is a sign of a model that does not only exhibit use cases with commonalities, but also one that *suffers* from repetition. These are the use cases we target with the Template Use Case pattern.

Identifying the invariants

Use cases suffering from repetition indicate some common behaviour. It is not always easy to identify the repeating behaviour because not all use cases will have a similar layout. If this is the case, it may be worth starting by refactoring the use cases involved to the point that they have similar layouts. Having done this, any sections that have identical or similar wording become suspect.

The following patterns are good indicators of repetition:

- Sentences or paragraphs where only the nouns are different.
- Sentences or paragraphs where only the verbs are different.

Having identified the variations and the commonalities, we can categorise the invariants:

- Where only the nouns differ, we have a process invariant.
- Where only the verbs differ, we have a data invariant.

The use cases with data invariants should not have been suffering from repetition and maintenance problems. One possibility is that the use cases go into too much detail about the data, hence repeat themselves. These use cases may benefit from a little more abstraction in this instance.

The use cases that exhibit process invariance, though, may benefit from the Template Use Case pattern.

Modelling the invariants

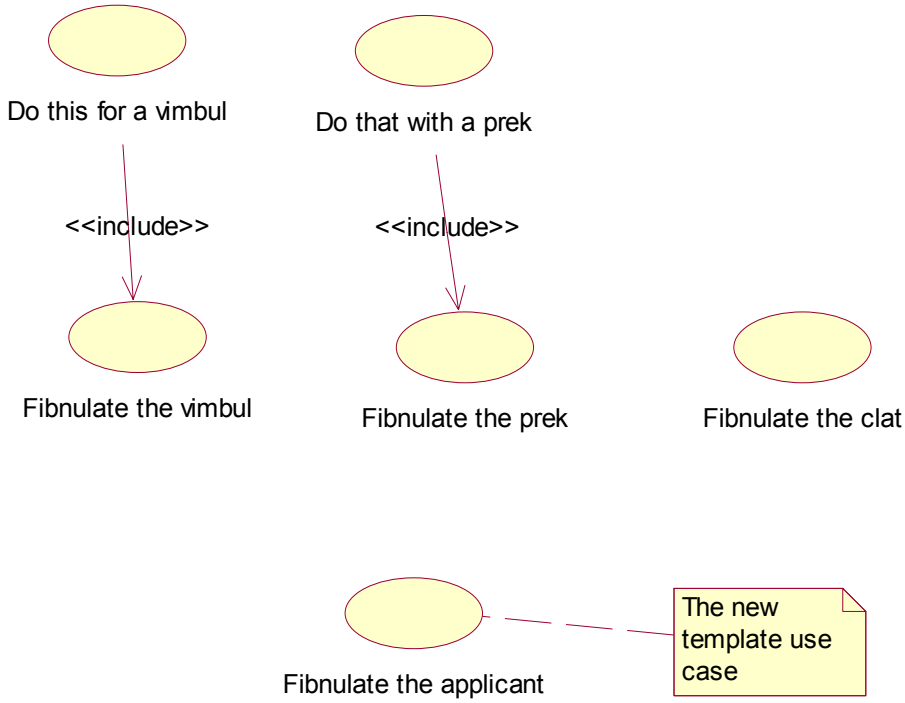
Having identified a process invariant, we can start modelling the invariant.

The process that is at the core of all the use cases, i.e. the common behaviour, is the invariant. This invariant will typically appear under two guises:

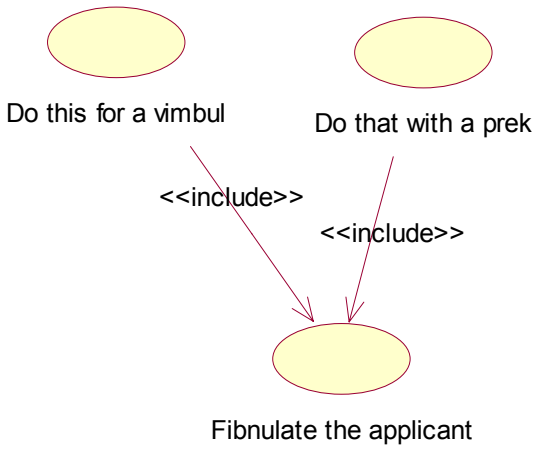
1. All the use cases have the same interactions in the same order throughout.
2. All the use cases have some common interactions as part of their flow, but there are other variations in the interactions.

In the first case, we can apply the pattern as it is.

1. Start a new use case. Give it a name that describes the general behaviour of all the use cases.



2. Copy the interactions from one of the use cases into the new use case. Any use case will do, as the interactions should be the same in all of them.
3. Replace the references to data with meaningful placeholder nouns. Use a different placeholder for each different type of data in the use case.
4. Replace the references to the old use cases with references to the new use case making the context (i.e. the variance) clear.



5. Model the variants, i.e. the data.

The template use case at the end would look something like this.

Fibnulate the applicant:

- System chooses the fiddle factor based on Application Type: high for complex application, low for simple application.
- The applicant supplies the required and the optional information.
- System works out the fibnulation factor and offers the outcome to the applicant.

The references to the template use case would be something like this:

Do this for a vimbul:

- …
- System validates the vimbul by fibnulation. Include {Fibnulate the applicant}
- …

Do that with a prek:

- …
- System decides the outcome of the prek by fibnulation. Include {Fibnulate the applicant}
- …

In the second case where not all the behaviour is identical, the first step is to factor out the common behaviour from the use cases. This takes the form of included use cases – one for each of the main use cases with shared behaviour. At this stage, the situation is identical to the first case (see the initial use case diagram). The rest of the model transformation work is identical.

Modelling the variants

Having modelled the invariant as a single use case, how do we show the differences?

The variants, i.e. the data, will in fact fit conveniently into a table. The placeholders that we used for the data in the use case text become the table's columns and the distinct nouns in the names of the original use cases become the rows.

Applicant type	Application type	Required	Optional	Outcome (fibnul. factor)
Vimbul	complex	name, address	DOB, id number	fiddle factor * 8
Prek	simple	name	address	fiddle factor * 3
Clat	simple	Name, address	previous addresses	fiddle factor * 1

This table can be part of the use case text when there is only one template use case. When there are several related template use cases with the same data variants, a document or spreadsheet attached to the use case package may be a better solution.

Other uses

This pattern applies to all use cases where the behaviour is the invariant. This includes use cases where the actors are different but the behaviour can be generalised. In this case, the template use case would be initiated by a generalised actor and the table's rows could be the child actors.

Issues

While the template use case helps reduce modelling effort, it may become a hindrance in projects where the deliveries are based on whole use cases. For project management purposes, it is generally much easier to track complete use cases than partial use cases. As the template use case subsumes a number of use cases, it cannot be tracked as a single unit.

We came across this situation in a project where we had automated project planning: We would pull the use cases from the model, and based on their priority and dependencies, we would automatically assign them to an iteration. We tracked progress based on use cases completed.

The solution was to keep the original use cases (i.e. the table rows) in the model but not to document them except for a reference to the template use case. These became the placeholders for the development effort.

Although this sounds like over-modelling, the ten minutes spent adding the use cases to the model can easily be justified by the management effort saved thanks to consistent automation.

One problem with a pattern like this is overzealous application. Because of the indirection it introduces to the model, I would recommend applying it only in cases where the invariant behaviour is a significant piece of functionality.

Summary

The Template Use Case Pattern can remove repetition in use case models by centralising invariant processes in a single use case or in a set of use cases. The technique consists of:

- Identifying the repetition.
- Moving the repeating interactions, i.e. the invariants, to a Template Use Case.
- Modelling the variants, i.e. the use cases containing the repetitions, as a table.

Acknowledgements and References

I first discovered this pattern in a project in 96. The paper was written for internal use at the time. When I decided to make it available publicly, I found out that it had been invented and re-invented independently by a good many people, so never bothered to publish it as there were few use case practitioners then. However, having repeatedly described the pattern a good half a dozen times and seen variations described in various forums, I assume that there are still enough people who might benefit from it. My thanks to Dave Fox and Scott Ambler for the latest prompts.

We used this pattern in a number of models to reduce the documentation effort for workflow use cases and crud-like behaviour.

