

CASE Tool Support for Use Cases

Hüseyin Angay

Karabash Ltd.

Member of the Appropriate Process Movement

<http://www.aptprocess.com>

Abstract

Use case modelling has long been viewed as a second class citizen next to object modelling. Some of this has to do with the relatively unglamorous image of requirements analysis compared with the design and implementation side of the business. Another reason for the unpopularity, however, is a lack of tool support for business and requirements analysis with use cases.

This paper lists a set of essential requirements for supporting analysis and development with use cases.

Copyright Note

This document resides online at

<http://www.aptprocess.com/whitepapers/CASEToolSupportForUseCases.pdf> and has been authored by Hüseyin Angay of Karabash Ltd. and the Appropriate Process Movement. It may be copied freely in part or in whole, with the restriction that anywhere using a copy of more than three paragraphs must include as reference the web address of its origin, as given above.

Contents

Abstract	1
Copyright Note	1
Contents	1
Introduction	2
Comparing use case and object modelling techniques	2
Current tool support for use cases	4
Essential tool support for use cases	5
Navigation	6
Filtering and searching	7

Categorisation.....	8
Validation	8
Synchronisation	9
Reorganisation	9
Simulation.....	10
Fundamental facilities	11
Summary	13
Acknowledgements and References.....	13
References	13

Introduction

One of the main contributors to the development of object modelling and its current popularity and success has been the tool support that has continuously improved over the last couple of decades.

Both in terms of popularity and support, use case modelling techniques have been lagging behind object modelling. This paper catalogues the kind of tool support that will be required to support use case modelling in order to make it as useful, effective and rigorous as object modelling.

Comparing use case and object modelling techniques

Use case modelling has been going through an evolution:

- It has been applied for years in niche projects without the rest of the world being even aware of the existence of the technique.
- It became accepted by a minority community and refined to the point that it could be put to a wider range of applications.
- It was forced onto a wider community of users who grudgingly took it on because their manager/client/colleague had heard good things about the technique.
- Gradually it acquired enough momentum that everybody had to "do use cases".
- Suddenly, everybody was a use case expert.

This, in fact, is pretty much the story with the leading edge of object modelling in '80s and early '90s. Given that use cases are following this pattern of application but trail by about 5-10 years, we can make a number of predictions about their application over the next few years based on what has already happened with object

modelling over the last decade. Namely, that the hype will calm down and a more sensible approach will emerge where not everything is a use case from screen designs to program specifications, for instance. Instead, they will be applied where they are most appropriate and much more successfully. This is a matter of time and mostly down to education and increasing experience in the industry.

The similarity does not end there. Along with the patterns of acceptance, the technique's heuristics also underwent a profound change. In the case of objects:

- They were applied as a better coding technique.
- Because their best selling points were inheritance, code with massive inheritance hierarchies became first desirable then the norm.
- This caused problems, so inheritance became a dirty word. We started to concentrate on flatter hierarchies.
- Complex hierarchies gradually crept back in. Now, class hierarchies in production code and models are much more complex than the ones considered unmanageable in early '90s.

Heuristics for the granularity and number of objects in models and code also follow the same pattern. That is, we first aimed for large numbers of small objects, found them to be unmanageable, gave up on the idea, then over time, accepted it as good practice, although we managed the complexity by encapsulating the smaller objects within components.

Now, we find use case modelling heading towards the same sort of crisis:

We have been building larger and more complex use case models for a few years now. However, as these models are starting to strain our ability to manage them, modelling with smaller and more numerous use cases and artefacts like use case dependencies are coming under fire from anyone who cares to comment. Yet, these play a significant role in making use cases more effective than other requirements analysis techniques, especially in the context of object technology implementations. So, how do we make sure that use case modelling evolves just like object modelling over the last decade and, at the same time, it avoids the pitfalls that made object technology stumble?

Education and experience will certainly make a difference. But, while skilled use case practitioners can be significantly more productive than those who are less experienced the techniques, they are still human: they can cope with only a limited amount of complexity. With time, practice and experience, we get better at organising models in our minds, but without external support, the majority of us hit a wall after a certain level of complexity beyond which our mental models cannot progress. I will not even bother to refer to the magic number seven, here. In the case of object modelling, this external support came in the shape of modelling tools that went beyond drawing utilities and provided more advanced facilities that went deeper into the semantics of the models. The same needs to happen for use cases.

Current tool support for use cases

The tool support for use case modelling is still inadequate. We can draw use case diagrams, attach properties and text to the use cases, organise them in packages and, if we are really lucky, establish links between our use cases and shall-style requirements items. When it comes to making sense of all this, we are on our own, because the CASE tools make no attempt to make sense of the use case models. The one concession that Rose makes, for instance, is to let us model collaborations, which are possible implementations of use cases; even those are there to make the object modelling and coding tasks easier, not to organise use cases themselves. Rose's integration with RequisitePro gives more facilities, but the operator still needs to supply and maintain most of the relationships manually.

The main reason that use case modelling has not been automated as heavily as object modelling yet is to do with the format of use cases – or a lack of format, to be more precise. The contents, in fact even the overall use case model itself, has been a free-for-all field until very recently. This has discouraged tool vendors from taking what might have become a bad investment decision: building facilities that they felt might not be used by the majority of modellers. Tools built in-house to manage use cases have also stayed in-house, because the different styles adopted in different enterprises, and even in different teams in the same enterprise, meant that most of these tools could not be used elsewhere.

UML has finally managed to come up with a consistent view of the use case model. The arguments over the difference between extending and included use cases are at last becoming the exception rather than the rule in my engagements with clients. Over the last year or

two, the style of the use case content promoted by a number of us over the years and finally published by Alistair Cockburn [Cockburn, 2001] has become the prevalent approach. Larry Constantine's essential use cases approach [Constantine, 1999], on the other hand, helped establish an accepted framework for the relationship between use cases and other artefacts such as screen designs and business rules. CASE tool suppliers, too, in their drive to establish their process frameworks, helped disseminate a number of practices that are becoming accepted principles for use case modelling. Given that the styles are not as divergent anymore, time may be ripe for introducing some tools.

Another argument for the introduction of tools is that they will encourage modellers to adopt styles that most suit the tools, assuming that the tools help them work more effectively. This should provide another impetus for convergence in approaches to use case modelling.

Essential tool support for use cases

So, what tools do we need for organising our use case models?

Tool support for object models comes with a number of facilities:

- **Navigation** helps the modeller to follow model elements along meaningful paths. Inheritance and association trees are now in just about every development environment in the case of class models.
- **Filtering and searching facilities** help reduce complexity of a model by displaying only the relevant model elements. We could for instance show only classes with certain types of operations and hide the rest.
- **Categorisation** helps organise model elements along different lines based on some inherent properties. This is similar to filtering, but the idea is to group the model elements rather than hide or show them.
- **Validation** is concerned with analysing the model and deciding whether it meets some criteria. For instance, can we map all the classes in a package to a relational model or do dependencies between packages reflect the associations between classes?
- **Synchronisation** ensures that related model elements are never out of synch with each other. This takes *validation* one step further and avoids invalid models by automatically keeping parts of the model consistent with each other. When an attribute's type is changed in the model, for instance, the

associated code may be automatically amended. Refactoring makes use of this facility to assist users change their code and models safely.

- **Reorganisation** takes *categorisation* one step further and makes changes to the model. For instance, we may change the package structure of the model or we may even go as far as creating new classes or attributes based on some properties, in order to automatically add data access objects to our model. Tools that assist code refactoring come under this umbrella, too.

Similar kinds of facilities will be needed if use case modelling is to improve enough to meet our increasingly complex modelling needs. A list of these facilities follows.

Navigation

CASE tools' inability to draw any meaning from use cases shows itself here. Use case text is a simple text field as far as the tool is concerned, and the attached documents are simply treated as BLOBs. Requirements management tools integrated with the CASE tools are a little more discerning as they will allow links to be established between documents, sections of text and model elements, but the links are still introduced and managed manually. For instance, the user may establish a link to another model element and, as a result, get hyperlinks to help them move between the two elements. There are two problems with this:

- (1) The requirements management tool generally enforces some configurable rules, so that an actor may not be linked to, say, a business rule. But as soon as we allow one use case to be linked to another, any use case can be linked to any other, without any validity checking.
- (2) The correspondence between the links managed within the requirements framework and the use case model must be managed manually: if we move the associations between use cases in the model, we must do the same in the requirements management tool.

Effective navigation will depend on the availability of the following facilities:

- Produce organised and navigable lists (for instance, a tree) of use cases, actors and other model elements as a result of questions such as: "Who interacts with this use case?", "Is it

included in other use cases?", "Which use cases extend this one?"

- Follow links from use case's text to other use cases, business rules and other associated model elements. At the very least, included and extended use cases and associated actors should be catered for.

Filtering and searching

Whilst CASE tools provide search facilities for model elements, most of these are limited to looking for keywords, locations and so on. Searches are not affected by the semantics of the model. Filtering is an extension of searching, in that it allows us to search for model elements, but then also lets us hide the model elements that we don't want to see. We have in the past tried to emulate this sort of facility by stereotyping and packaging use cases (we still do, in fact). We did produce some very interesting models in the process, although the interest will probably remain purely academic and certainly not aesthetic. And what to make of the numbers that we used to give to use cases (even to the point of naming them "028 – Apply for a loan") so that we could produce reports sorted according to our criteria instead of alphabetically (if we were lucky; some reporting tools would use the order in which the use cases were created as the sort key, too). Organising with stereotypes and packages helps with searching and gives us a manually maintained route to information hiding, but it also means that we introduce unnecessary information into the model just to persuade the CASE tool to let us do useful things with it – not to mention the hassle of having to maintain these structures and names. It would be much more useful if the CASE tool did this work based on the information we already put in the model. This would avoid both the effort spent on over-organising the model and the introduction of unnecessary information into the model.

CASE tools should provide at least these filtering and searching capabilities:

- List the use cases that fit certain criteria. Examples:
 - All the use cases that communicate with a certain actor.
 - Use cases with a certain priority (granted, this can be achieved by linking with a requirements management tool like RequisitePro).
- Draw diagrams for use cases that fit these criteria.

Categorisation

Filtering and searching are facilities for finding our way round the model quickly. Categorisation goes one step further and groups model elements to produce sub-models that serve a coherent purpose. The intention is to be able to treat that sub-model as a single piece for longer-term purposes such as work allocation or effort estimation.

Some examples of categorization:

- All the use cases that form a single work stream. [Angay, 2003].
- All use cases that can be fitted into the next iteration based on priority, dependencies, complexity and effort.

Validation

There is an increasing number of facilities in CASE tools and third party plug-ins for validating object models according to some heuristics. This could be anything as simple as ensuring that the class model may be mapped to a relational database or as complex as verifying the model against design heuristics (E.g. Do all one-way messages have associated call-backs? Do we rely on multiple inheritance?).

The scope for creating inconsistent use case models is, if anything, even greater. Whilst many of the class models are used to generate code, which then benefits from compiler checking and testing, use cases are reviewed only visually. Larger models are very difficult to validate visually.

I can already envisage the following kinds of validation:

- Validate the semantics of the model: Use case text and the associated model elements (including associations and dependencies) should agree; badly formed structures should be avoided (e.g. circular dependencies).
- Apply heuristics to use case models to enforce good style and usability – with the help of complexity metrics and so on.
- Allow new heuristics to be introduced to fit the special needs of the modellers.
- Provide metrics to show complexity, etc..

- Provide context-sensitive help and guidelines for the modeller in order to keep the model consistent, correct and usable.

Synchronisation

One issue with maintaining several interrelated model elements independently of each other is that inconsistencies creep into the model very quickly. Whilst validation can help point out these inconsistencies, it would be much better not to have those inconsistencies in the first place. For instance, when we change an association in one diagram, all CASE tools are clever enough to update that association in all the diagrams where it is displayed (mainly because the diagrams contain only a representation of an association stored as a single entity in the repository). Some CASE tools will also keep the class models in synch with the generated code. But what about the contents and relationships of use cases?

I would expect CASE tools to be clever enough to manage the following dependencies automatically:

- Use case text and:
 - Associations
 - Activity diagrams
 - Names of activities
 - Ordering of activities
 - Sequence and collaboration diagrams
 - Running commentary that links the use case text to the messages
 - Names of major objects
- Business and system use case models
- Use cases and associated business rules and requirements

Reorganisation

This is the ultimate aim of tooling in terms of modelling support. Reorganisation takes categorisation and synchronisation one step further and manages fundamental changes to the model. There are in fact two aspects to this:

- (1) *Production of significant chunks of a model by applying some assumptions to the existing parts of the model.*
Producing trivial mappings that can be consistently derived from one part of the model is not the intention. If this were the case, the derived parts of the model could be thrown

away as they could always be regenerated. The intention is to reduce the effort spent in straightforward but labour intensive tasks. One example would be a first stab at creating the system use cases from the existing business use cases. Another might be to synchronise two such models that had originally been produced independently.

- (2) *Refactoring*, i.e. managing the cascading changes that result from one or more related changes to the model.

Example:

When a use case associated with several others is changed, the CASE tool could take the modeller to every use case where it is included (and to the point where it is included in the text) so that they could verify the impact and make the necessary changes. The same would happen for all model elements that are effected by the change. The tool would take into account the cascading nature of some of the major changes: We change the behaviour of one included use case, which in turn effects the behaviour of a few others, which in turn affect the use cases related to them and so on. Nothing would be committed until the resulting changes to all the affected model elements had been made. If the change turned out to be troublesome (the impact introduced inconsistencies), there would be the option to carry out a complete or partial rollback and to have another run through the process. The commit aspect can be currently managed manually using a configuration management system (CMS), but the CMS follows the user's directions: The save-points and baselines depend on the modeller's judgement. As the magnitude of the impact may not always be obvious from the beginning, the modeller may do many smaller changes before a significant change that initially appeared to be insignificant. Rolling back to a previous version of the model would, in this case, cause the loss of all the other unrelated changes and create more work. So the CASE tool should allow context sensitive rollbacks (a multiple level undo facility, at the very least) independent of the CMS.

Simulation

I include this with some reservations, to the point that it's not even amongst the original categories listed earlier.

Simulating the behaviour of use cases can happen at several levels: stepping through the text, stepping through the associated sequence diagrams or, taking it one step further, stepping through the rest of the class model and showing the attributes that get changed and so on. I suspect not everyone would agree which of these levels or even any were relevant or useful.

Animating the sequence diagrams for use cases has already been done. Select Enterprise has been one example for the last five years or more, although I am not certain whether the resulting trade-offs (such as reversing the direction of «extend» associations) were worth it. I also get the feeling that activity diagrams will soon be animatable in some of the CASE tools. But these facilities are still gimmicks – i.e. features that can be implemented without a lot of effort or risk and get a tick in a feature comparison matrix. I remain unconvinced about their utility.

A much more productive first attempt might be to combine the text and associated model elements of a single flow in a use case chain into a single entity. This is much more useful in demonstrating the behaviour of a set of use cases. A simulation through the same chain, with appropriate questions at branching points could be a useful extension of this, but I am just as comfortable looking through a tree-like illustration of the chain's text. Any suggestions in this area would be appreciated.

Fundamental facilities

The facilities listed previously lead to a number of more fundamental requirements for the CASE tools, chiefly to do with the semantics of the contents of use case models. CASE tools understand some of the meaning locked in the use case models, mainly at pictorial level. We need to take the semantics beyond that of simple associations and packaging structures, especially since these are essential but less complex aspects of use cases. Once the semantics of the use case text and other related model elements are unlocked, the more complex modelling facilities will become available. The complex facilities I listed earlier may be introduced at plug-in level, but meeting the requirements I am about to list will not be as straightforward for third party developers. These should be a fundamental part of the CASE tools just as the semantics of, say, class models are.

- Users should be able to relate use case text to other model elements – e.g. highlight a textual reference to an included use case and select a use case from a list. The CASE tool may help here by selecting and sorting use cases, actors, etc. by similarity to the text highlighted.
- Even better, textual elements may be identified automatically. Some use case text will refer to other model elements, e.g. actors participating in a use case. There should be a way of identifying those and linking them without the user having to manually link the elements. This is most workable when modellers use conventions such as enclosing use case names in curly brackets –

given this hint, it should be straightforward to identify the connections and at least warn the modeller if there are no related model elements by that name.

- Facilities for quick textual inclusion of model elements that are already known to be related to the use case would be very useful, too.
- Here are some textual references within the use case body that should be linked to model elements:
 - Other use cases (including the nature of the association)
 - Actors
 - Classes, attributes and operations
 - Activities
- Other correspondences that should be tracked:
 - A use case line/paragraph and the annotations in a sequence diagram (so that the flow in the use case is always linked to the flow in the sequence diagram)
 - Post-conditions of one use case should be automatically linked to the pre-conditions of other use cases that mention the same state. An automatic dependency should exist between use cases thus linked.
- Automated line numbering support for use case text. Line numbers are more than a way of keeping use case text looking neat or programmatic (i.e. clever). They are useful when constructing the alternative flows and when referring other parts of a use case. Unfortunately, we cannot use the automatic numbering provided by the word processor: The numbers change when we move lines around and the references to them very quickly fall out of synch. The CASE tool should be able to manage the line numbers for use cases automatically and update all references to those lines when the numbers change. It is worth noting that references appear not only within a use case but within other use cases, too – we routinely write specialised use cases that mention the line numbers from the generic use case, for instance.

Summary

It is possible to make use case modelling more rigorous and more relevant to both the requirements analysis and the development process. If the CASE tools can help us make more sense of the semantics of a use case model and can manage those semantics, the additional rigour would not result in additional burden on modellers.

The facilities given here are neither exhaustive nor, I guess, all essential to everyone. Then again, once they become available, I would not be surprised if most modellers came to regard them as a natural part of the toolset.

I have customised Rational Rose on several occasions and Select Enterprise a handful of times in order to meet some of these requirements. As I get the chance, I will publish more of the scripts I have available. Some are more robust than others and none of them have proper user interfaces, but I hope they will stimulate some interest and, ultimately, some activity to produce something a little more usable.

Acknowledgements and References

This document is revised as necessary; comments are welcome. Thanks to the initial reviewers:

Thanks also to the following, who provided comments that improved the document:

References

Angay (2003)

Angay, Hüseyin (2003). *Use Case Chains*.
<http://www.aptprocess.com>

Cockburn (2001)

Cockburn, Alistair (2001). *Writing Effective Use Cases*. Addison Wesley

Constantine (2003)

Constantine, Larry L.; Lockwood, Lucy A.D. (1999). *Software For Use, A Practical Guide to the Models and Methods of Usage-Centered Design*. Addison Wesley