Using Enactable Models to Enhance Use Case Descriptions

Keith Phalp
ESERG
Bournemouth University, UK
kphalp@bmth.ac.uk

Karl Cox
Computer Science and Engineering
University of NSW, Australia
karlc@cse.unsw.edu.au

Abstract

Many tools developed for process modelling either model client business processes or the software development process itself. In both cases, benefits are to be found by using the model to highlight real process problems either of clients or developers. However, the modelling of client business processes allows a further opportunity for gain, where the intention is to build a system to provide support for the process being modelled. Although process models inform the requirements process, by providing clarity and understanding at the business modelling stage, the potential of such technology is often lost in the subsequent development phases.

The premise of the work described here is to use enactable state-based approaches, previously used successfully in business process modelling and simulation, to improve artefacts of requirements engineering, by providing enactable versions of use case descriptions. This allows for the kind of validation and checking so useful to business models. In particular, such models can be used to inform design, by providing rigorous scrutiny of the (low-level) details of use case behaviour.

The efficacy of this approach was gauged initially by producing enactable equivalents of use case descriptions using the existing process modelling language and tool RolEnact. However, industrial application also found that there was a mapping overhead and, hence, end users were reluctant to devote their time to producing enactable use cases without increased automation. This suggested a pressing need for tool support. That is, a use-case description tool which provided enaction capability, but without need for any further description. A prototype use case enaction tool is, therefore, introduced, along with a discussion of development issues and possible future directions.

1. Introduction

The rationale for considering the application of process technology to use case descriptions is that despite many years focus on software process, the requirements phase still seems somewhat neglected by practitioners. Glass [1] notes that many software projects fail due to poor or non-existent requirements processes. In the UK, Hall et al. [2] report that 48% of development problems are in the requirements phase. The potential cost of such problems is significant. Standish [3] estimated that "in 1995 American companies and government agencies will spend $81 billion for cancelled software projects" and $59 billion for those ('challenged' projects) that overrun in time or budget and which deliver (on average) only 61% of specified functionality.

One might wonder why this situation still occurs, particularly with a proliferation of methods purported to address the entire development process. For example, the Unified Modelling Language (UML) offers a host of diagram types for the software developer. However, support for requirements activities within the UML is provided primarily by one diagram type (and associated descriptions) the Use Case [4]. Though it has been suggested that objects are obviously identified in Use Cases [5], experience shows that this is not the case when considering design [6]. A number of other authors have also questioned whether Use Cases provide sufficiently comprehensible descriptions, not only for the description of requirements (e.g. [7, 8]) but also for moving towards design [9].

Jarke [10] elaborates upon this: The UML "hardly satisfies the demand for an adequate communications medium between users, developers, and other stakeholders." That is, the Use Case (and particularly the use case description) does not provide sufficient information to support meaningful dialogue between stakeholders. Maiden and Corrall [11] point out that "engineers rarely know… what the content and structure of … scenarios should be," (p2/1) because they lack

usage guidance. Hence, validation and revision of such descriptions is often difficult. Indeed, Anda and Jorgensen [12] state that the "lack of studies on Use Case Model understanding means that the guidelines and practices on how Use Cases should be described to ease their understandability… is highly subjective," (p.2). Despite the importance of Use Cases (being the only part of the UML geared to the requirements phase) Jarke [10] also notes that Use Cases and scenarios are hardly supported by present UML-oriented tools.

The L'Ecritoire tool [13] does not specifically address enaction; though it is concerned with the way descriptions are written. The SCORES tool focuses on verifying and validating the transition from use cases to class models [14]. The ScenarioPlus Toolkit [15] records scenarios as descriptions of requirements without consideration of enaction. Indeed, even Rational Rose™ barely supports use case descriptions, though other tools (e.g., [14]) may be used from Rose.

In contrast, many process-modelling tools, though often possessing impressive (enactable) functionality, do not consider the use case description, nor do they address requirements concerns. Thus, there appears to be a need for a use case tool that addresses two key issues, the representation of textual use case descriptions and also the formal enaction of subsequent descriptions as a validation of requirements.

Hence, there are two research problems to be addressed. Firstly, that the use case description itself, as it stands, is not adequate. Secondly, that tool support is also lacking. This paper examines the kind of information that use cases lack, and then suggests how tool support (and in particular the use of enactable models) addresses this issue. In suggesting this approach, the work borrows heavily from process modelling, which has used state-based paradigms to highlight such issues. In particular, exploratory work used the RolEnact business process modelling language [16] to provide models equivalent to given use case descriptions.

The rest of this paper is as follows. Section two examines the inherent weaknesses of use case descriptions, particularly with respect to their support for understanding the dependencies among events. Section three further examines the idea of dependencies and the use of enactable models to clarify such issues. Section four describes experiences with enactable models. These experiences suggested the need for an enactable use case tool; see section 5. Section six describes the issues and development of the use case enaction tool. Finally, section seven offers some conclusions and considers further work and unresolved issues.

2. Use Cases and dependencies

A number of authors have reported various problems both with use case diagrams and descriptions, e.g. [17, 18]. In particular, use case descriptions lack state-based information, and do not explicitly include any details about the dependencies of actions (except those at the start and finish of the whole description). Sutcliffe [19] suggests that an examination of dependencies in scenarios (and, hence, Use Cases [20]) can be used as a validation mechanism between requirements and specification and would be of use in subsequent design.

This paper argues further, that a full understanding of dependencies among actions is crucial to subsequent design and implementation. Hence, we now discuss the kind of issues that arise when dependencies are not fully and explicitly stated, and show the problems of finding such dependencies among use case description events. As an example, consider a generic use case.

SubjectA verb1 ObjectX
SubjectB verb2 ObjectY
SubjectC verb3 ObjectZ

Upon reading the description, one might wonder under what circumstances the final action (verb3) is able to occur. That is, is it dependent upon verb2 having taken place, or perhaps, verb1, or both or neither? It is usually taken that such information is clear from the text. However, as many will attest, such (implicit) assumptions in requirements are at best optimistic, and at worst dangerous [21].

Often, validation of such issues is important, particularly where the domain knowledge of the users may stop incorrect assumptions being made. Consider two (very similar) sporting examples, illustrated by the following use case description.

1 The match reached full-time
2 The referee blew his/her whistle
3 The ball crossed the goal-line
4 The goal was not given

Alternative Flow of Events

4a The goal was given

Consider the dependencies in this Use Case. It is normal to assume a roughly transitive dependency. That is, 3 follows from 2, which follows from 1. However, we may not be certain of this nor the dependencies. For example, let us suppose that our chosen sport is football

(that's soccer to some readers). We might wonder what line 4 (whether the goal is given) is dependent upon. In football the game ends at full-time, irrespective of whether the ball is still in play. Hence, the referee blowing their whistle (line 2) is dependent only upon line 1 having occurred (although the referee has to look at their watch to know this). At this point, the match is over. Therefore, if the ball crosses the line after this (line 3), then no goal is given.

A famous incident occurred in the 1978 World Cup in a match between Brazil and Sweden, where exact ordering had significant repercussions. With only seconds left a corner was given to Brazil and quickly taken. Only instants later, a goal appeared to have been scored and the referee blew his whistle (apparently) as the ball crossed the line. All commentators assumed that a goal had been scored. However, referee Clive Thomas claimed that he had blown for full-time (not to signify the goal) just before the ball had crossed the line. Hence, play had ended an instant before the goal had been scored. Despite much protest, Clive Thomas refused to allow the goal.

A similar incident occurred at the 2002 Commonwealth Games in Manchester, England in a field hockey match. With only seconds to go, a corner was taken and the ball crossed the goal line (event 3) after full-time (event 1), which had been signalled (event 2). However, in this case the goal did count, because in hockey the play from a short corner must be completed for the game to be over.

For these examples, the choice of which line 4 is used, main flow or alternative, is dependent upon the context. That is, the choice is dependent upon the correct interpretation of the rules of the game, and not just upon the order in which it is written. Therefore, the correct transition to event 4 (or alternative event 4) is not entirely apparent solely from the order of the use case description. It is the understanding of context and its impact on the dependencies that govern the movement from one event to the next. Understanding both context and description is necessary for the correct interpretation of the process – which may not be the order which the description alone implies.

These may appear trivial examples (though they caused much controversy at the time) but they serve to illustrate important points. Hence, in order to understand a problem domain, it is necessary to make such issues as explicit as possible, and to validate this understanding with end-users and domain experts. This means that communication with this non-technical audience is paramount, and hence that accessible modelling strategies may need to be employed to take account of this change of audience [22].

3. Consideration of dependencies and enaction

One way to force consideration of dependency and related design issues is to ask questions of the use case description by means of interrogation. This is becoming accepted in determining the accuracy of use case diagrams [23] and in checking scenarios against specifications [19].

However, a more formal approach (and, it is hoped, through tool support, more complete and less time-consuming) is to add (named) states to the description. This allows the modeller to formally consider triggers (or actions), which move the roles, actors or objects from one state to the next. This is particularly beneficial when considering synchronisation of independent objects. Finally, one might attempt a simulation (or a walk-through) of the possible actions and states. All of these may help to bring such issues to light. Experience suggests (both from process modelling [16, 24] and latterly our own) that enaction provides significant benefits in terms of understanding these issues.

Hence, in order to force consideration of dependencies the authors saw the need to move to the production of more formal (enactable) models that were equivalent to use case descriptions. The use of such (more formal) models would not only allow rigorous scrutiny of the descriptions, but also, through enaction capability, would allow for rapid, more complete validation. In order to test this idea, the existing process modelling notation RolEnact was utilised. RolEnact models were produced for each use case description, which explicitly forced the choice of pre and post states for each event of the Use Case. Rather than describe RolEnact the interested user is referred to original papers [16]. The following section describes experiences with these (RolEnact) and other enactable models.

4. Experiences with enactable models

In testing our ideas about use case enaction we used postgraduate software engineering students, typically undertaking larger group projects. Our experiences (over five years of using RolEnact based descriptions) suggested that models were relatively easy to produce (though with some overhead) and did help students to clarify issues. However, we realised that these student projects were not representative and therefore attempted this approach with an industrial collaborator [9].

The organisation concerned develops complex web-based applications for the online buying and selling of

stocks and shares and other financial packages in a highly competitive market. Each Customer application procedure is long and the details of the accounts opened are dependent upon the options selected by the Customer over the web. Therefore, it is imperative for the organisation to understand which options are selected and how their site reacts to that selection – different accounts require different information so correct dependencies are vital to the success of the product. The Customer application process was first modelled with Role Activity Diagrams [25]. These provided a high-level overview of the process. From these diagrams, finer-grained use cases were then derived to describe actor – system interactions at the application interface.

Other authors have also attempted similar approaches. For example, Kosters et al [14] mapped use cases the enhanced UML activity graphs (as part of the SCORES method), again with the idea of enhancing requirements validation and verification. They report that their enhanced validation led to benefits in product quality over a number of applications. However, they also note some scepticism by users, about the amount of effort required up front.

Although users (by the end of projects) may accept the benefits of such approaches, it was apparent that the increased modelling complexity (even if only perceived) was acting as a barrier to the acceptance of these kinds of technologies. For example, in our case, it was clear that production of a state-model (even in a simple language like RolEnact) was not an attractive proposition. That is, although they found the approach beneficial, the overhead of learning, what appeared to be a formal approach, was somewhat off-putting for developers. Hence, there was a need to allow users to gain this kind of functionality, without having to write (albeit simple) code, or map to other notations. This led to the realisation that there was an urgent need for 'simple to use' tool support.

5. Need for a use case enaction tool

As stated, early experience with enactable models of use case descriptions suggests that, though useful for validation, they were not always feasible for industrial application (mostly due to the perceived overheads involved in construction of the enactable models). This led to our move to develop tool support. The tool needed to allow the user to step through sequences of available events, using state based information to control their dependencies and thus which events were available at any given point. However, the tool also needed to produce this enaction directly from a normal use case description. That is, the tool should provide a use case editor, which will then allow the user an enaction capability without the need for further work.

The main problem with attempting such automation is that the basic use case does not contain sufficient data for one to 'know' which events are dependent upon others. Hence, the tool would need to allow the user to augment events, in order to make ordering explicit, or to change the ordering or dependencies. These issues, and our solutions to this dilemma, are now discussed with respect to development of a prototype 'use case enaction' tool.

6. Issues encountered in developing a use case enaction tool

An enaction tool aids the participation of both the customer and the requirements engineer in the validation process. In particular, the requirements engineer may easily show the customer the implications of the behaviour implied by use case descriptions. Indeed, this increased understanding on the part of both parties, and has shown to be one of the major benefits of enactable process models [22].

6.1. Default dependencies and enaction

In order to appeal to many development organisations it was felt that there should be a low-entry cost (in terms of understanding) for new tool users. Therefore, we wished to provide a basic level of functionality for minimal effort on the part of the user. Hence, for the first prototype the default dependencies implied by the use case description are assumed. That is, the normal ordering of events is used to produce a default dependency. The tool also displays the number of the event, which acts as a pre-condition in a final column. Hence, event 2, in the default case, is shown to be dependent upon event 1, and so on (see figure 1).

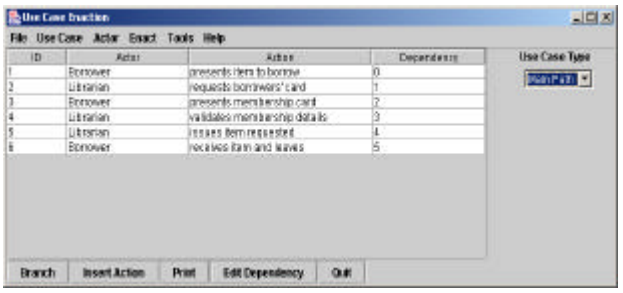

Figure 1. Actors, actions and dependencies shown within a description

The advantage of this approach is that it saves the user time, since (unless the implied sequence is incorrect) no further information is required. In addition, the non-technical user may still gain validation benefits, by stepping through use cases, and possibly revising them, without needing to delve into an understanding of states.

An example of an enaction step is given in figure 2 below, along with the description from which it was generated. This is taken from a variant of the classic ATM use case description [13].

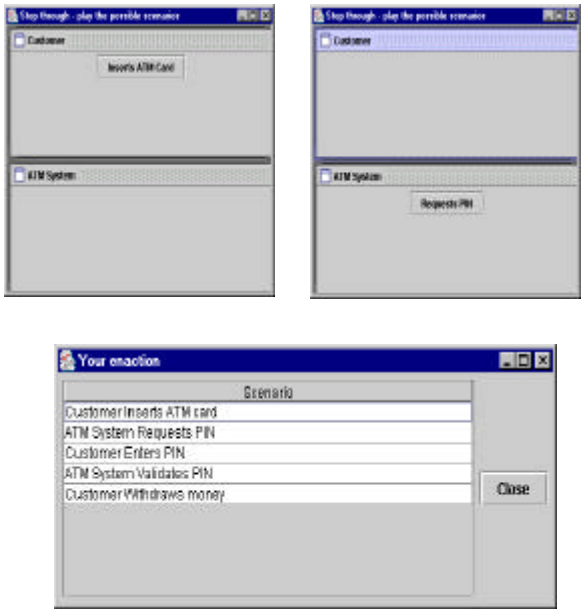

Figure 2. A simple enaction and related description

However, there are two possible disadvantages to this simple approach. Firstly, that the user can only control ordering (of enaction) by re-ordering the description. That is, the flexibility afforded by process modelling languages, such as RolEnact is lost. For example, more complex dependencies, where events depend on multiple circumstances, or where there are choices, concurrency and the like cannot be described. (Though of course these constructs cannot easily be accommodated within normal use case descriptions without exponentially increasing the complexity of the use case model.) Secondly, that the user is not explicitly forced to consider the dependencies for each event. The danger of making the enaction too simple to achieve is that the deep understanding gained by forcing a formal scrutiny of dependencies may be lost.

6.2. Levels of usage

The dilemma in providing a simple to use tool is that some of the enaction capability is compromised.

The mechanism proposed to deal with these 'default' dilemmas is for the initial description (and enaction) to assume the default ordering (see figure 1), but to allow the user to change the dependencies for each event when they wish to deviate from this default.

For example, consider the simple library scenario, where the librarian (user) is to issue the book to the customer (or 'Borrower'). [*Strictly speaking, one should not model the behaviour of the borrower for a use case specification. However, use cases are often used in this kind of domain modelling role*].

The default ordering would assume that event 4, 'Librarian validates membership details', is dependent upon only event 3 'borrower presents membership card'. However, it may be that the action is dependent on both presenting a card and on event 1, 'Borrower presents item to borrow'. That is, the librarian only moves on in the process, when they have both the book and the membership card. An example of this change of dependencies is shown in figure 3 (note the shorthand form of AND, which is shown as the intersection of 1 and 3).

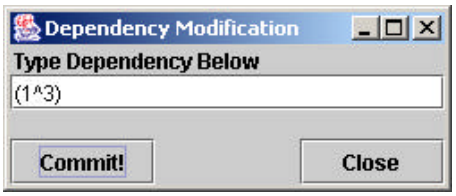

Figure 3. Changing dependencies

Other simple conjunctions of dependencies, such as logical OR, allow for choice, parallel threads and so on. In addition to changing dependencies, the user may also add further pre and post states for each event.

This two-tiered approach provides the flexibility and capability required for detailed enaction (across multiple use cases) when potentially vital events are examined, whilst also allowing the more straightforward use case to be enacted with minimal effort.

As can be seen, the tool allows two main levels of usage. At the first level the requirements engineer and the participating stakeholders may simply write the scenario as a simple sequence running from trigger through to completion. In this case, the default is that an action is simply dependent on the preceding action. However, at the next level more complex dependencies (pre and post

conditions) may also be added by making simple changes using the dependency modification features. These additional dependencies are typically within the single use case, but it may also be that an event in one use case is dependent upon another event within a different use case on the same project. (This need to consider multiple use cases is discussed in section 6.4).

6.3. Alternative paths

One of the main features of the use case description is that of describing alternatives to the main path. (Note that for implementation of enaction we consider exceptions and alternatives as the same functionality, even though we realise that semantically these are different).

Typically, an alternative (or exception) starts with a condition that makes it relevant, the pre-condition for that alternative. The alternate path might contain several steps describing what happens under that condition and ends with delivery of its goal or its abandonment.

Once the goal of the path has been achieved, or abandoned, then the extension post-condition must be true. For example, again consider a user borrowing books from a library. The behavioural branch that occurs when the borrowing limit is exceeded should be recorded with a pre-condition that the borrower has more books than allowed limit. However, the subsequent sequence of steps that occurs within this alternate path may also have multiple outcomes. For example, the goal of the alternative may be to achieve a correct amount of books, thereby returning to the main path, or may be abandonment of the use case, or may even be the initiation of another use case (perhaps to deal with fines). Hence, the user may wish to make both pre and post conditions for the path explicit. Figure 4, shows the simple edit box for adding pre and post conditions for an alternative path.

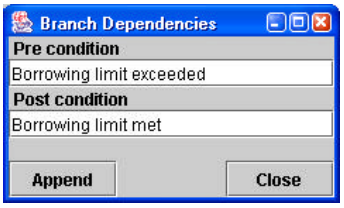

Figure 4. Adding conditions to alternative path

The handling of some conditions and system behaviour requires further domain knowledge and might mean further elicitation of requirements. Cockburn [26] argues that extensions are where the most interesting system requirements reside. The team members usually know the main success scenario. However, failure handling often uses business rules that the developers do not often know. Indeed, requirements engineers frequently have to research the correct system response and quite often that research introduces a new actor, a new use case or a new extension condition [26]. Hence, the tool regards all events and all paths as equal. That is, any path may itself have an alternative (or an extension). Equally, any path may also link to another use case (see below).

6.4. Multiple use cases

In the previous discussions, multiple interacting use cases have been suggested in two ways. Firstly, we note that the post-conditions for an action could come (as is usual) from another event within the current description, or from an event in another description. (Once again experience with process models shows the importance of this realisation for non-trivial processes.) Secondly, we note that any line in a use case (either within the main path or an alternative) may link to another use case.

Indeed, this reflects our development, in that we started with single (standalone) use cases and very soon realised that the potential for the tool was in aiding the more complex situation, which reflects real processes where aspects of one use case (or process) have an impact upon another.

However, there are many issues still to be resolved with our attempts to provide multiple use case support. For example, although from a requirements perspective each description should be regarded as equal, from an enaction perspective it seems that some hierarchy is helpful. Hence, we are still experimenting with different development approaches, and are in the process of 'trying out' these early efforts with current student cohorts.

6.5 Other opportunities (use case guidance)

The opportunity of developing a use case description tool allowed us to introduce support for writing guidelines. Such guidelines are typically based upon best practice, but also draw upon other foundations such as discourse processing, and have been shown to improve the quality of use case descriptions [27]. Again, our goal was to be able to incorporate such guidance without requiring significant overhead on the part of the user. In particular, from the very first prototype [28] the tool supported use case description guidelines (CP Rules [27]), by providing simple syntactic structures. The most obvious of these is the use of the subject verb object format (CP Structure 1). The tool also offers guidance to

the user about other best practices, for example the avoidance of past tense or passive voice by optional application of (an expandable) use case rules repository that flags known 'bad' or unrecognised words and grammars.

6.6 Experience so far

Our initial student-led experiences led us to realise that we needed to make improvements to the usability of the tool. For example, one of the problems raised was in editing or changing the placement of an alternative path on an existing description. We had assumed that if an alternative path were shown to be incorrect it would typically be deleted. We had not considered the possibility that a path might be internally consistent, but misplaced within the description. Students suggested that it would be better to provide functionality enabling users to simply move the alternative path elsewhere (in the Alternative Flows of Events sub-section). Closely related was the idea of re-organising already written descriptions (re-ordering) without rewriting the entire description from scratch (as described in 6.1) and the ability to add new events to the main scenario.

As with our previous experience using RolEnact models, these student experiences have proved useful in providing timely feedback on both the process (of tool usage) and the product. However, we are now attempting industrial exposure, and are currently collaborating with an organisation specialising in the production of software for the support of legal processes. (Indeed, previous experience suggests that the development of such products is enhanced by process-based requirements validation [29]).

.

7. Conclusions

This paper describes weaknesses in the 'normal' use case description, and suggests why such weaknesses are an area of concern. Typical use case descriptions do not consider dependencies between individual events. This means that both the sequence and dependencies among these events are implicit. However, further scrutiny of these dependencies often engenders greater understanding, both of the domain and the specific problem behaviour. This is of vital importance, not only to requirements validation but also to subsequent design.

One way to force rigorous scrutiny of dependencies is the production of formal (often state-based) models, but such models are not always well received by end users. This means that a major opportunity afforded by validation, to gain understanding from domain experts, may be lost.

A solution to this dilemma is the use of enactable models, which provide both rigour (in production) and communication (when being enacted). The authors report on experience of using enactable models (a commonly used technique in process modelling) both with students and industry. In brief, it was found that existing enaction approaches (such as RolEnact) proved valuable for validating use case descriptions but were perceived (by collaborators) as too much of an overhead for industrial acceptance.

Hence, a use case tool was introduced, which provides an enaction capability from a normal textual use case description. The issues encountered in attempting this tool support are described, as are proposed resolutions. For example, it was found necessary to produce two levels of usage; where a default dependency is assumed, which can then be edited and augmented by the user.

Experience so far is limited, however, this paper sets out a need to provide support for both the production and analysis of the use case description. Given the importance of requirements validation, and the potential benefits of enaction, the acceptance of such support promises many benefits. However, developers have traditionally been reluctant to increase the proportion of effort devoted to requirements activities.

The authors contend that tool-sets of this nature, which provide an enaction capability 'for minimal effort', increase the possibility of such industrial take-up.

7.1 Further work

We are currently developing the tool with feedback from students, other members of faculty and industrial collaborators. Thus far, we can enact for single descriptions and are experimenting with different strategies to enact multiple threads through a number of descriptions. A further idea is to support the situation where there are a number of related scenarios (differing only by small numbers of choices). It is thought that these will need to be bound together, so that all possibilities are accessible from the main flow. For example, the user might want to load all (or chosen) scenarios with a simple button press. However, there remain a number of unresolved research questions, such as those outlined below, and we believe that these suggest a need for further research:

- Does the increased capability offered by dependencies enhance or overcomplicate descriptions?

We are of the opinion that this is an enhancement that will be of importance in design as much as in requirements validation. In any case, the user can use the basic enaction functionality when it is not necessary to explore dependencies in such detail.

- Will the inclusion of use case writing guidelines restrict the flexibility offered by enaction?

Entirely unconstrained use case structures can lead to ambiguity and are ultimately difficult to validate (and later implement). Thus some guidance makes sense. So long as the enaction tool can interpret the description event (and should be able to do so if the event is formatted to meet a simple guideline construct [30]) then this should not be an issue.

- Does the template approach to structuring use cases fit more naturally with tool support?

It would seem sensible to provide users with a use case template. This gives them the opportunity to consider all potentially relevant features within the context they are in. This should not affect the enactability of the description.

- Will requirements volatility make dependency mapping unmanageable?

Although requirements are notoriously changeable (and this has an impact upon subsequent use case descriptions [30]), we plan to assess how this volatility impacts upon the dependencies within and between the descriptions. It is our contention that this impact will be minimal since dependencies are at a lower abstraction than the requirements. However, fundamental requirements changes will naturally affect the nature of the use case descriptions (as many descriptions may simply be thrown away or sidelined for a later release).

- Do users really require models that consider dependencies across use cases, or does the restriction to consideration within a use case provide a partitioning of understanding?

We believe that enaction is of benefit and that consideration of dependencies is a powerful verification and validation tool. Detailed dependency consideration should also be importance because designers and coders can explore design alternatives to enable a better structuring of the underlying design and implementation.

8. References

[1] Glass, R., *Software Runaways*, Prentice Hall, Harlow, 1998.

[2] T. Hall, Beecham, S. and A. Rainer, "Requirements problems in twelve software companies: an empirical analysis", *Proceedings of the 6th International Conference on Empirical Assessment in Software Engineering*, Keele, Keele University, 8-10 April 2002.

[3] Standish Group, "The Standish Group Report: Chaos", http://www.scs.carleton.ca/~beau/PM/Standish-Report.html, 1995.

[4] Jacobson, I., Christerson, M., Jonsson, P. and G. Overgaard, *Object-Oriented Software Engineering: A Use Case Driven Approach*, Addison-Wesley, Wokingham, 1992.

[5] Jacobson, I., 2001. Introduction. In: Armour, F. and G. Miller, *Advanced Use Case Modelling*, Addison-Wesley, Harlow, 2001, ppxiii-xiv.

[6] E. Insfran, Pastor, O. and R. Wieringa, "Requirements engineering-based conceptual modelling", *Requirements Engineering Journal*, 7 (2), 2002, pp. 61-72.

[7] Graham, I., *Requirements Engineering and Rapid Development*, Addison-Wesley, Harlow, 1998.

[8] Jackson, M., *Problem Frames*, Addison-Wesley, Harlow, 2001.

[9] K. Phalp and K. Cox, "Guiding use case driven requirements elicitation and analysis", In: Wang, Y., Patel, S. and R. Johnston (eds), *7th International Conference on Object-Oriented Information Systems*, LNCS, Springer-Verlag, Calgary, 27-29 August 2001, pp. 329-332.

[10] M. Jarke, "CREWS: towards systematic usage of scenarios, use cases and scenes", *Wirtschaftinformatik 99*, Saarbrucken, Springer Aktuell, 3-5 March 1999. CREWS Report Series 99-02: http://sunsite.informatik.rwth-achen.de/CREWS/reports.htm.

[11] N. Maiden and D. Corrall, "Scenario-driven systems engineering", *IEE Seminar on Scenarios through the System Lifecycle*, London, 7 December 2000, IEE (Ref 00/138), pp. 2/1-2/3.

[12] B. Anda and M. Jorgensen, "Understanding use case models" *Beg, Borrow or Steal: Using Multidisciplinary Approaches to Software Engineering Research, Proceedings ICSE 2000 Workshop*, Limerick, June 5 2000.

[13] C. Achour, "Guiding scenario authoring", In: *8th European-Japanese Conference on Information Modelling and Knowledge Bases*, Vammala, Finland, 26-29 May 1998, pp. 181-200.

[14] G. Kosters, H-W. Six and M Winter, "Coupling Use Cases and Class Models as a Means for Validation and Verification of Requirements Specification", *Requirements Engineering Journal*, 6, 2001 pp. 3-17.

[15] I. Alexander, *Scenario Plus*, www.scenarioplus.org.uk

[16] K. Phalp, P. Henderson, R. Walters and G. Abeysinghe, "RolEnact: role-based enactable models of business processes", *Information and Software Technology*, 40, 1998, pp. 123-133.

[17] Rosenberg, D. with K. Scott, *Use Case Driven Object Modelling with UML: A Practical Approach*, Addison-Wesley, Harlow, 1999.

[18] K. Cox and K. Phalp, "A case study implementing the Unified Modeling Language use case notation version 1.3", In: Opdahl, A., Pohl, K. and M. Rossi, (eds), *6th International Workshop on Requirements Engineering: Foundation for Software Quality*, Stockholm, 5-6 June 2000, Essen, Essener Informatik Beitrage, pp. 69-78.

[19] A. Sutcliffe, "Scenario-based requirements analysis", *Requirements Engineering Journal*, 3, 1998, pp. 48-65.

[20] I. Alexander, "Scenarios in systems engineering – a range of techniques for engineering better systems", *IEE Seminar on*

Scenarios in the System Lifecycle, London, 7 December 2000, IEE (Ref 00/138), pp. 1/1-1/6.
[21] Gause, D. and G. Weinberg, *Exploring Requirements: Quality Before Design*, Dorset House Publishing, New York, 1989.
[22] K.T. Phalp, "The CAP Framework for Business Process Modelling", *Information and Software Technology*, 40 (13) 1998, pp. 731-744.
[23] Armour, F. and G. Miller, *Advanced Use Case Modelling*, Addison-Wesley, Harlow, 2001.
[24] T. Kalito, D. Partridge, K. Phalp, D. Raffo and J. Ramil, "Working Group Report: ICSE'99 Workshop on Empirical Studies of Software Development and Evolution", *Empirical Software Engineering Journal*, Vol. 4, No. 4, December 1999.
[25] Ould, M., *Business Processes: Modelling and Analysis for Reengineering and Improvement*, Wiley, Chichester, 1995.
[26] Cockburn, A., *Writing Effective Use Cases*, Addison-Wesley, Harlow, 2001.
[27] K. Phalp and K. Cox, "Supporting communicability with use case guidelines: an empirical study", *6th International Conference on Empirical Assessment in Software Engineering*, Keele, Keele University, 8-10 April 2002.
[28] Hageseter, T., *The Production of an Enactable Use Case Tool*, Masters Thesis, Bournemouth University, November 2002.
[29] Mitchell, M., *The Development of a Conveyancing System*, Masters Thesis, Bournemouth University, November 2002.
[30] Adolph, S., P. Bramble, A. Cockburn, and A. Pols, *Patterns for Effective Use Cases*, Addison-Wesley, 2003.