Goal Sketching With Activity Diagrams

Kenneth Boness
University of Reading,
Berkshire, RG6 6AY UK
k.d.boness@reading.ac.uk

Rachel Harrison
Stratton Edge Consulting Ltd,
GL7 2LS, UK
rachel.harrison@strattonedge.com

Abstract

Goal orientation is acknowledged as an important paradigm in requirements engineering. The structure of a goal-responsibility model provides opportunities for appraising the intention of a development. Creating a suitable model under agile constraints (time, incompleteness and catching up after an initial burst of creativity) can be challenging. Here we propose a marriage of UML activity diagrams with goal sketching in order to facilitate the production of goal-responsibility models under these constraints.

1. Introduction

A goal and responsibility model represents the stakeholders' hopes (their intention) for a system-to-be that will operate in an expected environment, in fulfilment of a contract. Such models (e.g. Figure 1) are widely used in goal-based requirements engineering such as the KAOS approach [i].

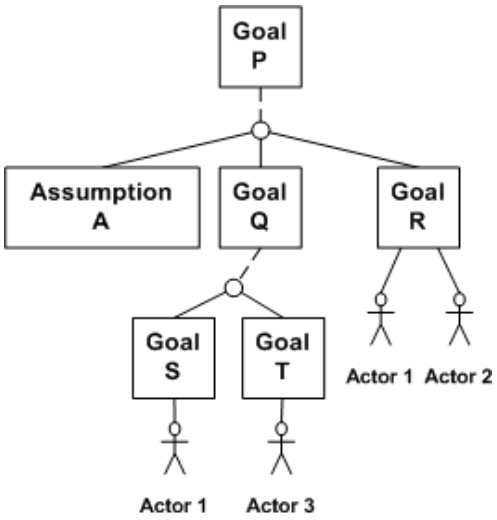

Figure 1. Goal-Responsibility Model

If the model is *structurally complete* all objectives are ultimately satisfied by actors of the system-to-be. The behaviour (and other qualities) that must be instantiated is described only at the leaves of the model instead of being distributed across the model. This is helpful when appraising the model for its feasibility, adequacy [ii] and testability. It will help to ensure that we "create realistic expectations in the minds of the stakeholders" [iii].

However the prerequisite to such benefits is to be able to create a goal-responsibility model very early-on in a real project; probably as a crude sketch to be evolved as understanding increases.

This need has motivated our interest in goal sketching [iv] based on natural language with AND entailment (shown by the circles in Figure 1). Such sketched models can be made easy for stakeholders (e.g. managers and customers). to understand provided that care is taken over the level of detail shown. However poor construction of the model can be counterproductive.

We have found that analysts have a tendency to be very uncertain about the appropriate goal formulation (refinement argument). This often results in a poor model. This seems particularly evident in the following circumstances:-

1. Elicitation: where stakeholders are inclined to express their requirements as partial, hypothetical, functional designs.
2. In backlog driven projects: which, after a few sprints, reach the point where there are inadequate specifications for regression and acceptance testing; often result of an initial creative burst of *ad hoc* development by a team of experts.

Observing that activity diagrams are good at representing functionality and processes, we posed the research question: can they offer a practical compliment to goal sketching? This paper explores this question.

After a brief summary of the meaning structural completeness section 2 explains how activity diagrams used with a set of guidelines can be used to build goal responsibility models as complements to activity

diagrams. A key move in representing activity diagrams in goal-responsibility models is the introduction of a class of goal that imposes the logic that glues the activities together. It is presented as a goal such as "Impose process X". In brief reports of two early industrial examples we illustrate the successful creation of dual views and the role of this particular class of goal. We see qualitatively the value of having one viewpoint to focus on understanding functionality and the other to appraise intention. The paper concludes with discussions on related work and further work.

2. Structural Completeness

Figure 1 represents system behaviour and qualities *intended* to be developed. It follows the keep all objectives satisfied paradigm and thus resembles a KAOS goal and responsibility analysis. It shows how all objectives (eg the single goal P) are refined in a stepwise manner into sub goals which are entailed by a set of actors in the system. By enacting their responsibilities for the goals S, T and R the system actors satisfy goal P by entailment provided that assumption A holds. When all objectives are satisfied by responsible actors the model is said to be *structurally-complete*.

3. Method

The idea is to express functional requirements through activity diagrams and then transform them into goal-responsibility models. This makes a dual representation possible; the activity diagrams can be used as intuitive viewpoints for stakeholders who more naturally think in virtual designs and the goal-responsibility model can be used for more general representation to direct inspections and the appraisal of feasibility and adequacy.

3.1 Guidelines

In order to maximise compatibility between the activity diagrams and goal sketching we used the following guidelines:-

1. Activities are behaviours and so they should be described as goals [v].
2. Every activity in an activity diagram must be either (a) supported by a use-case specification with fully assigned responsibilities to system actors, or (b) represent a nested (decomposition) activity.
3. A *structurally complete* activity diagram has all actors traceable by use-cases; if necessary traced through nested diagrams.
4. The UML activity diagram notation should be kept to a simple subset initially allowing more precise notation to be added later as appropriate.

3.2 Complimentary Viewpoints

In order to introduce the method and show how the two complimentary viewpoints are produced we will assume that there is an intention that involves developing a product (PM) to operate in a particular application domain [vi] such as illustrated in Figure 2.

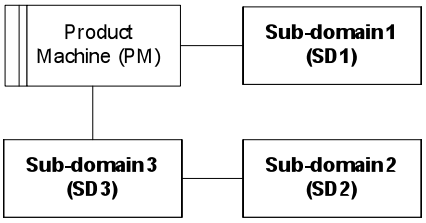

Figure 2. Domain Model

Let O stand for the primary objective (such as managing patients' records in a particular medical environment). Suppose that O would be satisfied by the activities expressed in the UML activity diagram shown as Figure 3.

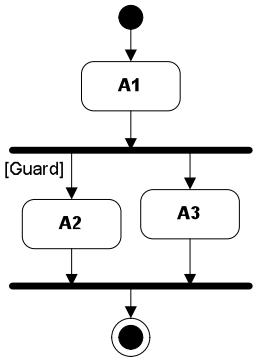

Figure 3. Activity Diagram

As O concerns behaviour it can be represented by a goal statement G0. Similarly the activities can be regarded as having goals G1, G2, G3 for A1, A2 and A3 respectively. Given that our wish is to work with goal-responsibility models we might propose that the set {G1,G2,G3} entail G0 as in Figure 4.

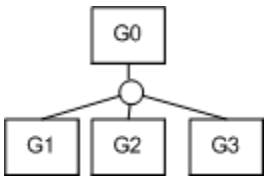

Figure 4. Proposed entailment of G0.

The set is necessary but does not account for the logical context of A1, A2 and A3 (guard, fork, join etc) included in Figure 3. It is therefore not sufficient. It can be made so by adding an enforcing goal such as *impose process A* as in Figure 5. Formally we have:-

$$\{\text{Impose process O, G1,G2,G3}\} \models \text{G0} \qquad (1)$$

Where semantic entailment ($\models$) is used because the goals are specified in natural language.

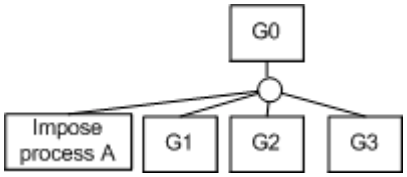

Figure 5. Goal refinement of G0

If any of the leaf goals in Figure 5 have associated use-cases defining how they are satisfied by responsible actors (which must be drawn from Figure 1) they should be added. Figure 6 shows an example where we have assumed that the actor PM is responsible for the goal *impose process A* etc.

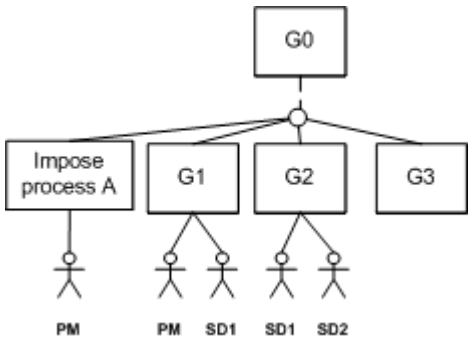

Figure 6. Incomplete Goal-Responsibility Model

Figure 6 is structurally-incomplete because G3 needs further refinement. Suppose that G3 is satisfied by the activity diagram A3 shown in Figure 7.

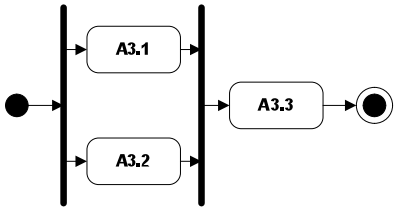

Figure 7. Activity Diagram for G3 (A3).

Then treating G3 in the same way as above for G0 and assuming responsibilities for the new goals (G3.1,G3.2 and G3.3) the structurally complete goal-responsibility model Figure 8 is produced.

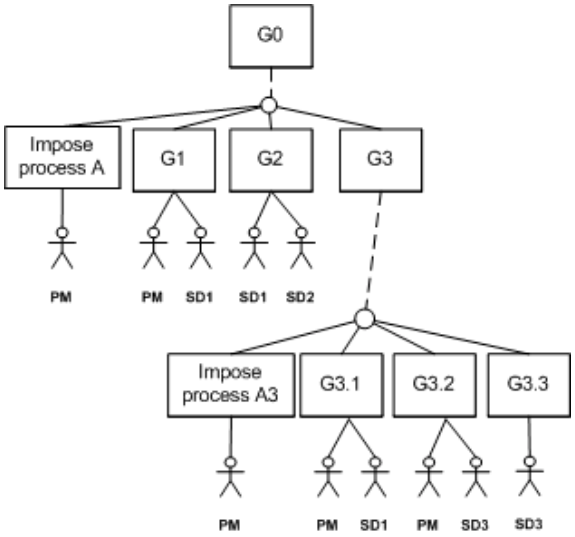

Figure 8. Structurally Complete Goal-Responsibility Model

3.3 Specifying the Leaves

Normally every leaf goal (behaviour) is specified with a use-case. Hence most leaves will have multiple actors responsible for them. Cockburn's casual and fully dressed forms can be used depending on the rigour that is appropriate. Alternatives include IEEE capabilities [vii] and problem frames [viii] depending on circumstances..

3.4 Composition

Sections 3.2 and 3.3 have shown in principle how to create the complimentary viewpoints of activity diagrams and goal-responsibility models. However goal refinement involves decomposition. Hence although Figure 8 is structurally complete the re-composition of the responsibilities must be considered. This is especially so for the "Impose Process X" class

of responsibility introduced in this paper which will in general impact on all its sibling goals and their descendants. For example in Figure 8 "Impose Process A" must be composed with G3.1 etc as well as G1-G3. The designer and the tester may discover significant complexities.

The non-functional requirements must also be included in a full representation of a real intention. These have not been the subject of this paper but can be can be added to the goal-responsibility model as further goal nodes and the resulting cross-cutting responsibilities can be composed just as any other responsibility [iv].

4. Industrial Examples

We introduce two industrial examples: one each for the two circumstances outlined in the introduction.

4.1 Example A: Clinical Audit Product

The first case is the specification of a small scale development of three person months. The intention was to provide a care audit product to be used in UK medical practices.

An initial analysis of stakeholders' concerns informally captured an eclectic mixture of constraints, abstract goals and design prejudices as surrogates for requirements; just as predicted in [ix]. After expressing the overarching goals, assumptions and constraints in a high level goal model it was evident immediately which parts of the intention were normative (well known to the developer and stakeholder community and needing little elaboration) and which parts were radical (needing detailed evaluation). This analysis took only a few hours to complete but made early appraisal possible and immediately paid back by allowing the limited staff resources to focus on what matters in negotiation ("creating expectations" see introduction) and difficulties in engineering.

The radical functional goals required consultation with typical users. This led to the agreed activity diagrams such as shown in Figure 9. Note:-

- Nested activity diagrams are needed to refine "Prepare sponsor…." And "Send to depository…." (as indicated by the UML convention of a fork symbol).
- All other activities are the equivalents of goal leaves and have use-cases expressing the required behaviour of the system actors.
- The guards are not rigorous but were sufficient for the purpose of discussing requirements.
- The guards include factory (or installation) configuration conditions as well as run-time conditions.

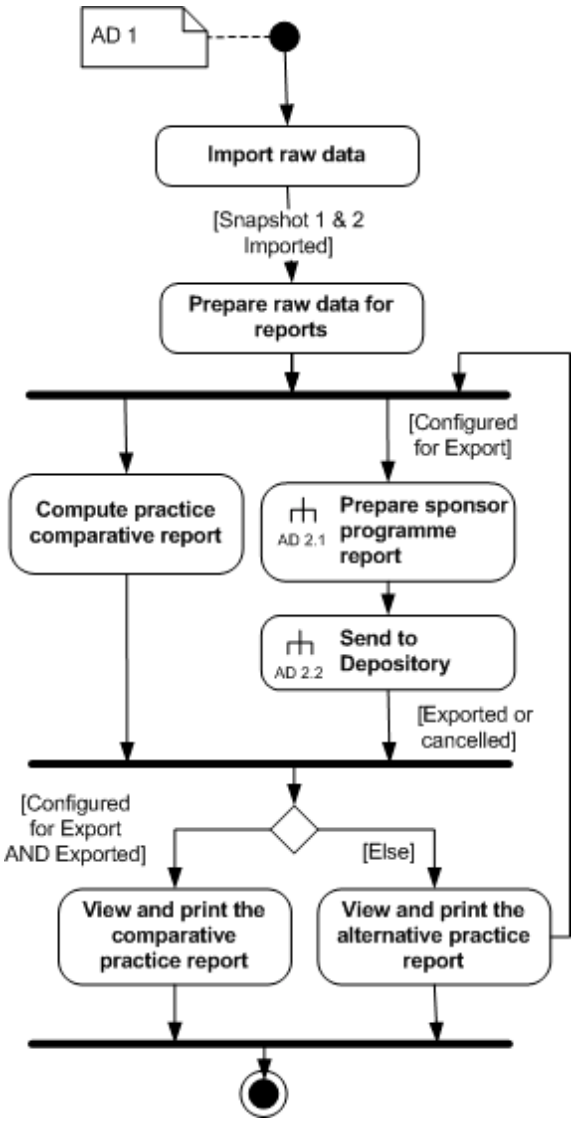

Figure 9. Primary activity diagram.

A detail from the corresponding goal-responsibility model is shown in Figure 10. Note:-

- The top goal is not a root goal as there are higher goals above it.
- Goal "Impose Activity A1" Enforces the logical flow/states of the activity diagram of Figure 9. A second "Impose Activity…" Enforces the logic of a sub activity AD2.1. These are sub-levelled in a manner corresponding with the activity diagram nesting.
- Limited space has necessitated leaving out some of the activities shown in Figure 9; "Prepare raw…", "View and Print…"are omitted.

- All leaf goals have actors (Tool is the software product; MIQUEST is an agent used in the UK for extracting medical information from medical systems). The behaviours are defined by use-cases; some use-cases have multiple actors.

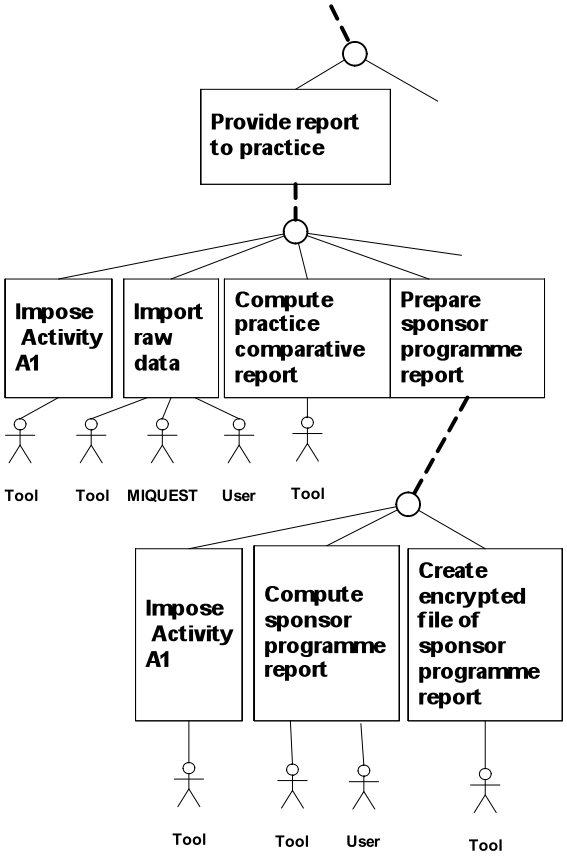

Figure 10. The partial goal-responsibility model

In Figure 10 all activity is at the leaves. The project manager, tester and developer have to be sure that all the leaves are feasible. The analyst used the goal-responsibility model refinement argument to inspect the adequacy of the intention. The stakeholders could be talked through the model but the real value was to aid the analyst with a disciplined method of appraisal. This dual method has been used on several projects in the company since with success judged on willingness to re-use the method in the commercial environment.

4.2 Example B: Enterprise Product

This case shows the application of the method to the second circumstance listed in the introduction. Whereas example A is essentially an *a priori* problem this one concerns the *a posteriori* problem of acceptance testing. The example is a large scale development of a database security tool by a UK based company. Development has followed the Scrum (agile) iterative process to turn an initial concept into a product to satisfy a wide range of customers and environments.

The development has been ongoing for three years and the product is now on the market yet still has significant potential for evolution. The development team has grown over the three years from five to twelve engineers; early on one of these being a part-time tester and recently three are full time testers. A backlog driven sprint methodology was adopted with an onus on testing to demonstrate at the end of each sprint that the new backlog items had been accomplished and that none of the old backlog items had inappropriately regressed. The number of backlog items engineered so far is over 3000. Like many backlog and iterative style developments the individual backlog items are not always pure requirements. They include desirable activities such as refactoring and non-behavioural changes. Consistent with good agile style the management have encouraged adequacy and feasibility as guiding principles when planning every sprint. They have always valued predictability of accomplishment over quantity of aspiration in their sprints.

The problem is that the backlog is insufficient as a resource to design regression and sprint acceptance; the sum of backlog tickets does not adequately reveal the coherent and holistic experience warranted for the product. This seems to be a common jeopardy with long term backlog based developments.

The problem was addressed by using the methods described in section 3 to reverse-engineer a model as a basis for planning regression tests. The activity diagram viewpoint has brought a uniform understanding of product behaviour shared across the company. The goal model has afforded a disciplined approach to test coverage analysis; this has created a trend of increasing incompletion faults at the end of each sprint and other latent faults that would otherwise only arise to compromise downstream sprints.

The reverse engineered representation involves one overview activity diagram and only two levels of nested activities. Across all diagrams there is a total of approx 60 leaf activities. Every one of these has a use-case which in turn has a set of test cases so that percentage test coverage can now be considered. Whilst presently some are run manually there is a programme to fully automate them.

A satisfying discovery was that the recognition of the "Impose Process X" class of responsibility revealed significant areas of product behaviour that were most vulnerable to error but had not been strongly recognised in testing based on original backlog entries.

This problem has provided an examination of the methods described here in terms of scalability and

utility for a large and complex product. The product has a complex array of configurations and versions.

In addition to this *a posteriori* application of the method there is now interest in using the models to allow better investigation of incremental change impact analysis which as anticipated in [x] is an increasingly significant problem.

5. Related Work

There is little direct mention of activity diagrams and goal oriented requirements engineering in the literature although in [xi] there is an implied association in the relationship of activity diagrams and use-cases.

The idea of writing use-cases to elicit and organise functional requirements is a good practice guideline [xvii] and is developed strongly in [v,xii]. The work reported here involves use-cases as part of a different tactic: Activity diagrams serve to elicit and organise functional requirements leaving use-cases to define leaf goals and their responsibilities. It is distinct also from the practice of associating use-cases with goals as espoused in [v]; there the goal hierarchy harbours responsibilities at all levels .

An interesting contrast to our approach is reported in [xiii]. There the emphasis is on using goal refinement as a means to discover use-cases as leaf behaviours. This is a contrast as the very point of our use of activity diagrams is due to our experience that eliciting functional goals is itself a problem.

More formal developments involve tactics with scenarios and UML. For example in [xiv] scenarios are used for their narrative, concrete and informal style of description to elicit abstract declarative specifications. In [xv] goal elicitation is guided by the use of scenarios. In [xvi] a UML profile provides industrial professionals with a familiar UML access KAOS model. We believe that such approaches are very significant and important but may be too formal for regular industrial practice (particularly agile practice). The goal sketching approach espoused here may be regarded as a preliminary survey to such methods and accordingly some formal bridging may be possible.

6. Discussion and Further Work

Good heuristics should help people to draw useful graphs rapidly. Traditional advice of embracing frequent negotiation with spiral emergence [xvii] and decomposing width before depth [viii,xviii] is applicable but more help is needed. The work reported here endeavours to make contribution in this direction.

The method reported here is work in progress yet has already has proved practical enough to be useful in the reported cases. Whilst quantitative gains are not reported here it is the case that their continued use against a demanding commercial background suggests practicality and usefulness.

The method has overlaps with the work of the use-case and more formal methods communities. We expect to develop the method to complement these endeavours; especially to provide appraisal viewpoints. In this way the method offers complimentary support rather than simply offering another requirements analysis technique.

The keys this complementary relationship are: (1) showing all responsibilities at the leaves and (2) the technique of introducing the *impose X goal* (e.g. see Figure 10). The second of these hides, but does not discard, complex control or process regimes. It presents them for appraisal. Activity diagrams could be substituted by other modelling artefacts such as state diagrams, use-case goal models or policy statements.

It might be observed that these complementary views contrast an ordered sequence of activities with an un-ordered set of goals. But this would be to overlook the power of the *impose X*. Indeed this became very important in the second of the reported examples where the acceptance tests had to focus on the glue between the activities.

In the practical examples presented the activity ‘guard’ is used to manage configuration as well as run-time conditions. This has opened possibilities for managing the common practical problem of portable software products: they have to be specified and tested against multiple configurations. We aim to explore this further.

7. Conclusions

Aiming to accelerate the process of goal sketching we have introduced a method of working with activity diagrams in tandem with goal-responsibility modelling. In this complimentary use the activity diagrams help the formulation of goal sketches for functional requirements and configuration permutation whilst the goal-responsibility models provide the basis of disciplined appraisal. The work is at an early stage but has at least passed the test of application to real world problems. There are possibilities for generalisation of the method that are to be investigated with the expectation that other structured and UML modelling methods can be incorporated.

8. Acknowledgements

Richard Olearczyk, Sean O’Mahoney of OSKIS Informatics and Dr Steve Moyle, James Wilson and Peter Guillebaud.

9. References

[i] Dardenne, A., Lamsweerde A., van, and Fikas, S., "Goal-Directed Requirements Acquisition", Science of Computer Programming Vol. 20, pp. 3-50, North Holland., 1993, pp. 3-50.

[ii] Boness, K., Harrison, R., Finkelstein, A., "A lightweight technique for assessing risk in requirements analysis", Software, IET, 2008, Volume: 2, Issue: 1 pp. 46-57.

[iii] Nevo, D., and Wade, M., "How to avoid disappointment by design", Communications of the ACM, 2007, Vol 50, No.4 .

[iv] Boness, K., and Harrison, R, "Goal Sketching: Towards Agile Requirements Engineering", Proceedings of the International Conference on Software Engineering Advances, ICSEA 2007, pp. 71-76

[v] Cockburn, A., *Writing Effective Use Cases*, Addison-Wesley, Boston, 2001.

[vi] Jackson, M., *Requirements Specifications – A lexicon of practice, principles and prejudices*, Addison-Wesley, Harlow. 1995.

[vii] *Guidelines for writing system requirements specifications*, IEEE Std 1223-1998.

[viii] Jackson, M., *Problem Frames*, Michael Jackson, Addison Wesley, Harlow. 2001

[ix] Alexander, I., Maiden, N., Sc*enarios, Stories and Use Cases*, John Wiley & Sons, Chichester. 2004.

[x] Rajlich, V., "Changing the paradigm of software engineering", Comm. of the ACM, Vol 49, No.8, 2006.

[xi] Ambler, S., *The Object Primer*, Cambridge University Press, 2004.

[xii] Alexander, I, and Stevens, R., *Writing Better Requirements*, Addison-Wesley, Harlow, 2002.

[xiii] Robinson, W., and Elofson, N., "Goal Directed Analysis with Use Cases", Journal of Object Technology, vol 3, no. 5, pp. 125-14. 22004,

[xiv] Lamsweerde, A., van, and Willemet, L. "Inferring declarative Requirements from Operational Scenarios", IEEE Trans on Software Engineering, Special Edition on Scenario Management. 1998.

[xv] Rolland, C., and Souveyet, C., et. al., "Guiding Modelling using Scenarios", IEEE Transactions on Software Engineering 24(12), pp.1055-1219. 1998.

[xvi] Heaven, W., and Finkelstein, A., "UML profile to support requirements engineering with KAOS", IEE Proceedings -Volume 151, Issue 1, 9, pp. 10 – 27. 2004.

[xvii] Sommerville, I., and Sawyer, P., *Requirements Engineering – a good practice guide*, Wiley, Chichester. 1997.

[xviii] Adolph , S, and Bramble, P., *Patterns for effective Use Cases,* Addison-Wesley, 2003