GOAL SKETCHING AND ACTIVITY DIAGRAMS

Kenneth Boness\(^1\), Rachel Harrison\(^2\)

\(^1\)Department of Computer Science and Informatics, The University of Reading, Reading, Berkshire, RG6 6AY, UK
\(^2\)Stratton Stratton Edge Consulting, GL7 2LS, UK

k.d.boness@reading.ac.uk, rachel.harrison@strattonedge.com

Keywords: Goal Oriented Requirements Engineering, Activity Diagrams, Goal Sketching, Natural Language Requirements, Agile

Abstract: Goal orientation is 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 report on an investigation into the 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 stakeholders’ hopes for a system-to-be, which will operate in an expected environment, in fulfillment of a contract. It represents their intention and can represent the rationale and understanding of the problem to be solved along with a set of domain assumptions and the stated requirements for a system. Models of this type are widely used in goal-based requirements engineering and most notably in the KAOS approach (Darimont, Lamsweerde and Fikas 1993, Lamsweerde 2001).

If the model is structurally complete all objectives are ultimately satisfied by actors of the system-to-be; as illustrated in Figure 1. A key quality is that all behaviour to be instantiated is described completely at the leaves of the structure – not distributed across the model as, for example, in some use case goal oriented methods. This makes them excellent for stakeholders to appraise the intention for its feasibility and adequacy (Boness, Harrison and Finkelstein 2008) and its testability.

Hence we may address the concern of (Nevo and Wade 2007) to “create realistic expectations in the minds of the stakeholders”; crucial in this agile world. However a prerequisite to this approach is that a model is available very early-on; probably as a crude sketch to be evolved as understanding increases. This need motivates our interest in goal sketching (Boness and Harrison 2007).

![Figure 1: Goal-Responsibility Model](image-url)
Goal sketching is based on natural language goal descriptions and uses the principle of AND entailment (shown by the circles in Figure 1) which is readily explained to lay stakeholders (such as managers and customers). The resulting goal-responsibility model can be made easy for such stakeholders to understand provided that care is taken over the level of detail shown. However poor construction of the model can be counterproductive. We find that with semantic entailment poor construction, or difficulties in formulation, is often associated with functional requirements in the following circumstances:-

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

Observing that activity diagrams are good at representing functionality requirements we posed the research question: can they offer a practical complement to goal sketching?

After introducing structurally complete goal-responsibility models this paper proceeds to explain how activity diagrams used with a set of guidelines can be used to build goal responsibility models. This gives the added value of providing two complimentary viewpoints: one to represent intended system behaviour and the other to facilitate disciplined appraisal of the intention. The paper provides two industrial examples as illustrations matching the circumstances described above and concludes with a discussion.

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 (Cockburn 2001)
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 Dual Viewpoints

Suppose we are developing a product for some application domain as shown in Figure 2. Then the intended software defines a product machine (PM) (Jackson 1995).
Suppose the intended purpose of the product in the application domain is to satisfy a particular high level objective (for example to manage patients’ records in medical practice environments). Suppose also that this objective (O) would be satisfied by the activities expressed in the UML activity diagram A shown as Figure 3.

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.

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:

\[
\{\text{Impose process O, G1,G2,G3}\} \models G0 \quad (1)
\]

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

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 a hypothetical example where we have assumed that the actor PM is responsible for the goal impose process A etc.

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:

\[
\{\text{Impose process O, G1,G2,G3}\} \models G0 \quad (1)
\]

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

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.
3.3 Composition

Although Figure 8 is structurally complete it has been derived by a process of decomposition. In principle all responsibilities need to be composed for any implementation of these requirements to be acceptable. However in general not all compositions are significant; i.e. they do not imply any complication required in the design. On the other hand some will be significant and may even harbour conflicts or further composition concerns (Jackson 2001). We suggested managing composition by the use of composition tags or introducing supplementary composition tables; the latter method is to be preferred when there are many responsibility compositions to annotate (Boness & Harrison 2007).

In Figure 9 the simpler method of using responsibility tags is used. All responsibilities significantly affected by impose process A are tagged “<ACTA>” and the responsibilities affected by impose process A3 are tagged <ACTA3>. Actor SD1 has not been tagged as an example of insignificant composition. Actor SD3 has been tagged twice (ACTA and ACTA3) as an example to show that nested activity diagrams lead to cumulative compositions.

3.4 Composing NFRs

In the goal sketching method non-functional requirements, including constraints, can be expressed as nodes of a goal-responsibility model and then made effective by composing them with goal responsibilities. For example a constraint C such as the user interface must be of type X can be expressed by adding a composition tag (say <UI>) to all responsibilities that concern the user interface. Figure 10 shows the responsibilities of Figure 9 that are affected. Note no actor for C is specified as it is a constraint and not a behaviour.

3.5 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. Sometimes a capability (IEEE 1998) or a problem frame (Jackson 1995, 2001) may be used instead.

Leaf constraints are specified in the manner recommended for well formed requirements as in (IEEE 1998).
4 INDUSTRIAL EXAMPLES

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

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 (Alexander and Maiden 2004). 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 11.

- 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.

A detail from the corresponding goal-responsibility model is shown in Figure 12. Note:-

- The top goal is not a root goal there are higher goals above it.
- Goal “Impose Activity A1” Enforces the logical flow/states of the activity diagram of Figure 11. 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.
- Not all of the activities are included to save space.
- Composition tags have been omitted in the interests of clarity in the limited space.
- 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 11: Primary activity diagram.](image-url)
stakeholders could be talked through the model but the real value was to aid the analyst with a disciplined method of appraisal. The activity diagrams were used to help the stakeholders negotiate the intended behaviour. This dual method has been used on several projects in the company since with success judged on willingness to re-use the method in a stern commercial environment.

Figure 12: The partial goal-responsibility model

Example B: Enterprise Product

Whereas example A is essentially an a priori problem this one is an a posteri problem; that is, it concerns requirements in 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 was a part-time tester but recently three engineers became 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.

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.

The reverse engineered representation involves one overview activity diagram and only two levels of nested activities. Across all diagrams there is a total of approximately 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.

In addition to this a posteri 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 (Rajlich 2006)) 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 (Ambler 2004) there is an implied association in the relationship of activity diagrams and use cases.

The idea of using use cases to elicit requirements is a good practice guideline (Sommerville and Sawyer 1997). It is developed strongly in (Cockburn 2001, Alexander and Stevens 2002). There are clear similarities but the current idea is distinct in the choice of the activity diagram, not the use case, as the principal organiser of functional analysis. Use cases are important here only as the specifications for leaf goals so that they can be assigned to actors. In Cockburn’s usage this is not necessary and high level goals may have responsibilities.

The practice that use cases are defined at the leaves of the goal hierarchy is reported in (Robinson and Elofsun 2004). There the authors take the position of using goal refinement to discover an artefact (use cases) whereas we are using an artefact (activity diagrams) to help identify operational goals. There is an interesting conflict of view between the idea that goal refinement is demonstrably straightforward and the idea that it benefits by support.

More formal developments involve tactics with scenarios and UML. For example in (Lamsweerde and Willemet 1998) scenarios are used for their narrative, concrete and informal style of description to elicit abstract declarative specifications. In (Rolland and Souveyet 1998) goal elicitation is guided by the use of scenarios. In (Heaven and Finkelstein 2004) 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 (Sommerville and Sawyer 1997) and decomposing width before depth (Jackson 2001, Adolph and Bramble 2003) is clearly important but more help for practitioners is needed. The work reported here endeavours to make a contribution in this direction.

The method reported here is work in progress yet has already been tested in the reported cases. Whilst quantitative gains are not reported here it is the case that the method has continued in use despite a demanding commercial background and this suggests it is satisfying practicality and usefulness criteria.

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

The keys to this complementary existence are: (1) showing all responsibilities at the leaves and (2) the technique of introducing the impose X goal (e.g. see Figure 12). 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.

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: such products 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 complementary use the activity diagrams help the formulation of goal sketches for functional requirements and configuration permutations 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 methods and UML modelling methods can be incorporated.
ACKNOWLEDGEMENTS

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

REFERENCES

(Boness, K., Harrison, R., Finkelstein, A., 2008), A lightweight technique for assessing risk in requirements analysis, Boness, K., Finkelstein, A., Harrison, R., IET proceedings, accepted for publication 2008.
(Cockburn, A., 2001, Writing Effective Use Cases, Addison-Wesley,1st edition, Boston.)
(Heaven, W., and Finkelstein, A., 2004,) UML profile to support requirements engineering with KAOS.