Bridging the Requirements Gap: Policies, Goals and Domains

A.G. Sutcliffe & N.A.M. Maiden

Department of Business Computing, City University,
Northampton Square, London EC1V 0HB, U.K.

Abstract: *A model for requirements engineering is described which uses a taxonomy of goal-types to guide further analysis. Goals are classified according to the desired system state described in requirements statements. Heuristics then prompt further description of functions according to each goal class. Other analyses encourage expansion of goal statements into specification of objects agents, activity and information processes. These link functional decomposition of requirements to object oriented modelling. Implications of the model and supporting tools are briefly reviewed.*

1: Introduction

One of the central problems in developing software specifications is how to transform informal linguistic statements of requirements into more formal expressions of software solutions for those requirements. Typically requirements are stated in language and commonly referred to as policies, goals, aims, objectives and so on. Such statements can only be reasoned about by systems capable of natural language understanding which are limited by their lack of domain knowledge. This paper suggests one approach to the analysis of such linguistic statements towards more formal models amenable to reasoning and ultimately proof in the software engineering tradition.

The central concept is to conceptualise goals as expression of future states which a problem solution has to achieve. The heritage for this approach comes from [1] which proposed successive decomposition of linguistically expressed goals until they could be described as state expressions. Another interesting approach was demonstrated [2] who examined the decomposition of policy statements and how contradictions in such statements are uncovered by successive refinement towards formal expression. Other relevant influences are argumentation structure models of decisions and design rationales which account for goals and their interaction [3]. However, all these approaches say little about the process of how goal decomposition influences development of software specifications. It is this angle that we wish to
explore.

2: Policy-goal model

This model supports the first three activities in the requirements engineering process: problem/policy investigation, policy/goal refinement and goal analysis. Goals are decomposed into successive levels of detail to form a tree of arbitrary depth. Lower level goals support achievement of higher level policy. Most parent nodes will have 'And' relations with the child nodes, as two or more sub goals will support the achievement of an upper level goal, however, there may be occasions when "Or" relations are required for alternatives. The model of decomposition helps structure the problem space by suggesting that policy is refined to identify the functions necessary for their achievement. Goal trees can be constructed to an arbitrary depth. Description of individual goals is matter of individual judgement about the problem in hand, what we intend to provide is a means of structuring the thought process of requirements analysis. First three levels of goal or policy requirements statements are distinguished:

i) policy level describes statements of what should be done. Statements at this level (see figure 1) do not contain any information about how the policy may be achieved, for example “improve customer satisfaction’. No particular constraints are placed on the expression of goals at this level;

ii) functional goal level has linguistic expressions containing some information about what may be done to achieve goals at the policy level, for example “improve satisfaction by answering customer queries promptly’. Further relationship types may be added to show goal conflicts such as ‘inhibits, promotes, enables’ to create an argumentation structure [3,4], however, this is not the main aim of our work. Instead we focus on analysing the goals themselves;

iii) domain goal level. At this level goal descriptions are refined by three treatments (see figure 1):

(a) analysis of opportunities for automatable processes to achieve goals. This treatment considers whether goal needs further analysis for specification of an automatable process or whether a management decision is required. Labels are attached to each leaf in the goal tree to label the implication for the next phase in analysis as:

- management implication, referred for a policy decision. These goals are not amenable to automation, instead a decision is required about resources, for example “Hire more staff to improve customer service”;
- operational implication, requiring some change to procedures carried out by people;
- opportunity for automated support, an information system could be introduced to implement a required function, or an existing computer system could be improved to meet the need.

Judgement about whether a process should be automated or not is taken in consultation with users. The main point is to identify where further definition of requirements can be made and to connect requirements definition in the System world of software engineering with the Usage world of human computer interaction, following the perspective of [3].

(b) templates are used to encourage addition of facts linking the functional view of aims and purpose to an incipient model of the domain in terms of objects, agents and information processes etc. The templates for refining goal expressions are:
Goal = action phrase, object by agent
(for example, help reader find items by search-facility)
Goal = action phrase, object with agent/resource
(for example, buy more titles with reserve-budget)

(c) goals, combined with non functional requirements, are expressed as a state which system has to achieve, for example "‘all customers’ queries must be answered within 3 hours of receipt, i.e. the system must never be in a state where queries older than 3 hours exist.". The leaves of the tree should all be expressible in terms of states. This necessitates a goal classification as detailed in the following section.

3: Goal analysis

We argue that goals fall into six classes and that those classes have utility in terms of their implications for further elaboration of requirements. To illustrate the contention with a library system, the goal types are as follows:

3.1: Positive state goals

These goals indicate states which must be achieved, for example “all books must be recalled within 7 days of due date’. In this case the library system must ensure that all books are returned within 7 days:-
Books [in-lib] = Books [in-lib, return-date - borrow-date >7 days]
Does Not exist: Books [out-lib return-date - borrow-date >7 days]

Positive state goals suggest specification of further functionality to ensure that the correct state is achieved and, furthermore, that the incorrect state is avoided. When this goal type is detected the following heuristics are used for requirements refinement:

Heuristic (A1): specify the necessary process to achieve the goal state, in the above case possibly a recall procedure;

Heuristic (A2): specify the process to detect exceptions and take corrective action, if necessary; in the above case a periodic sample of readers’ books outside the library to detect overdues.

3.2: Negative state goals

In this case the state expressed is to be avoided, goals are expressed as invariants or constraints on system behaviour, for example "All book loans can not be renewed more than twice". This implies that the system must prevent multiple renewals:

Book [renew number =< 3]

Negative states, as with positive states indicate specification refinement, however, the emphasis is on control procedures to ensure a state does not happen. Ultimately negative and position states are isomers of each other. A positive state can be achieved by preventing negative states occurring. Ultimately it is a matter of user preference whether requirements are stated positively or negatively. As before requirements refinement heuristics are suggested:

Heuristic (B1): specify the control test or process necessary to prevent the illegal state from occurring.

3.3: Alternative state goals

Two or more states are possible but the choice of which state applies depends on system input. In this case the choice of the state is delayed until run time when further information is available, for example "If the book is more than 7 days overdue then the reader is fined, unless the reader can prove that return of the book was impossible". Two states and a decision make up this goal and analysis of the selection starts the process of specifying how the goal is to be achieved by a method:

States: Readers-with-overdues = Fined / Readers-with-excuses Not Fined
If book [due-date >7 days] Then
Either Reader-penalty = fine
Or
If Reader-excuse = accepted Then
Do nothing

The heuristics for these goals urge the requirements engineer to consider the decision making and the possible consequences:

Heuristic (C1): refine the conditions of the test to determine which state applies;
Heuristic (C2): specify source of data for decision making;
Heuristic (C3): add default decisions if no data is available.

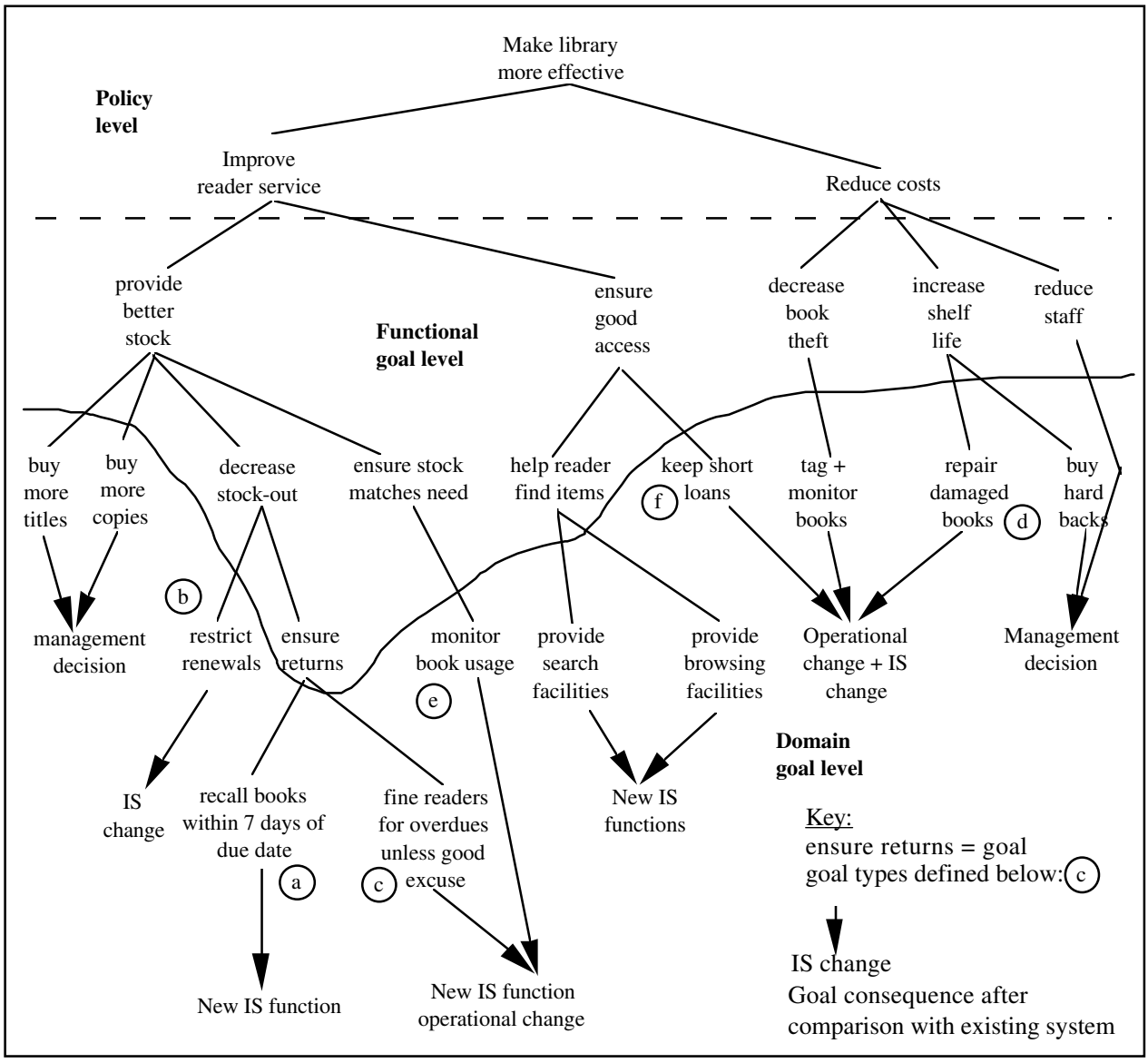

Figure 1 - goal hierarchy with policy, functional and domain goals (goal types: a=positive state goals; b=negative state goals; c=alternative state goals; d=exception repair goals; e=feedback goals)

3.4: Exception-repair goals

In these cases nothing can be done about the state an object achieves, even though it is unsatisfactory and therefore must be corrected in some way, for example "damaged books must be removed from stock and sent for repair". The goal is a sub type of positive goal states in that the object state is unsatisfactory and some corrective action must be taken. The sub type is distinguished according to assumptions about the inevitability of change; if an object will naturally arrive in a degraded state through natural wear and tear, then an goal of repair activity is necessary. Requirements refinement for these goals needs to address the monitoring process to detect unsatisfactory states and the corrective action:

Heuristic (D1): investigate the monitoring process for detecting the unsatisfactory states;

Heuristic (D2): specify the process for corrective action.

3.5: Feedback goals

These are associated with a desired state and a range of exceptions which can be tolerated. The system has to monitor the homiostasis of one or more usually a population of objects and take corrective action towards some predefined norm, for example "The books in stock should have an inter-loan interval of no less than 7 days and no more than 6 months in any 12 month period, otherwise corrective action should be taken." In this case the desire is to maintain a stock of books with a usage rate in a certain range, and if any departure from this range is detected corrective action is taken such as buying more copies if the book is popular or alternatively removing it to archive. This type is a variant of positive goal states; however, the goal is stated is stated as a target with a tolerance range, a non functional requirement, and action is necessary to return the system to the set target. The heuristics encourage analysis of two processes:

Heuristic (E1): analyse monitoring process;
Heuristic (E2): investigate correction process.

3.6: Mixed state goals

Mixed state goals require the system to maintains some balance between two states. These goals are another sub type of selected goal states except that it is a population of objects which are controlled. Two sub states may exist, and the distribution of instances between the sub states has to be achieved, for example " the proportion of books on short loan and read-only use in the library should be proportional to the demand for use" . This goal implies that criteria should be stated for determining proportionality between the two sub states. The state to be achieved depends on an event frequency:

If Book-demand ratio short-loans : in-lib = x:y Then
Ratio Book [short-loan] : Book [read-only in-lib] = x:y

Heuristics advise that the conditions for determining the state are investigated:

Heuristic (F1): investigate monitoring process for event capture;
Heuristic (F2): elaborate conditions for determining goal state selection;
Heuristic (F3): specify process for change balance of sub states in population.

So far six goal classes have been described and others may be added as our investigations proceed. The effect of this classification is to suggest requirements analysis in certain directions according to the class of goal.

Our final aim is to link the above goal analysis to generic models of systems which are embodied in the domain theory [5,6]. Briefly the theory proposes classes of domain models which are have a basis in object-state semantics. If the goal classes can be matched to the appropriate domain model and the model describes a suitable state to achieve the goal then the domain knowledge can be reused to further elaborate the specification.

4: Conclusions

The process model of goal refinement in requirements analysis offers a systematic means of typing linguistic statements in a manner than suggests the appropriate further analysis. While this classification requires judgement, the framework outlined above does provide a 'tool for thought' to help guiding the requirements engineer. The classes extend the taxonomy proposed by [1,7] and add requirements analysis heuristics, although the link to formal semantics has not been addressed. Instead we propose linking goal analysis to reusable domain knowledge implement this in an intelligent analyst-assistant tool for requirements engineering.

References

1. Dardenne A., Fickas S., & van Lamsweerde A., 1991, 'Goal directed concept acquisition in requirements elicitation', Proceedings 6th International Workshop on Software Specification and Design, IEEE Computer Society Press 14-30.
2. Feather M.S., 1993, 'Requirements reconnoitring at the juncture of domain and instance', Proceedings 1st IEEE symposium on Requirements Engineering IEEE Computer Society Press, 73-76.
3. Jarke, M., Bubenko, J., Rolland C., Sutcliffe A.G., & Vassiliou Y., 1993, 'Theories underlying requirements Engineering: An overview of NATURE at genesis', Proceedings 1st IEEE Symposium on Requirements Engineering, IEEE Computer Society Press, 19-31.

4. Conklin J & Begeman M.L., 1988, 'A hypertext tool for exploratory policy discussion'. ACM Transactions on Office Information Systems 6(4), 140-151.
5. Maiden N.A.M. & Sutcliffe A.G., 1992, 'Exploiting reusable specifications through analogy, Communications of the ACM 35(4), 55-64.
6. Maiden N.A.M. & Sutcliffe A.G., 1993, 'Requirements Engineering by example: an empirical study', Proceedings of IEEE Symposium on Requirements Engineering, IEEE Computer Society Press, 104-112.
7. Dardenne A., van Lamsweerde A. & Fickas S., 1993, 'Goal directed requirements acquisition', Science of Computer Programming 20, 3-50.