Extending Knowledge Engineering to Requirements Engineering from Multiple Perspectives

Debbie Richards Tim Menzies

AI Department, School of Computer Science and Engineering,
The University of New South Wales, Kensington
{debbier, timm}@cse.unsw.edu.au

Abstract: If the costs of requirements engineering (RE) are prohibitive, then RE will rarely be applied. We present an RE strategy designed to handle conflicting perspectives that is an extension to current KE techniques. We instantiate this approach in the context of formal concept analysis (FCA) and ripple-down-rules (RDR). FCA was used to build explanatory T-boxes from performance A-boxes created by an RDR system. This approach is applicable to any representation which can be mapped into a decision table. However certain representations offered advantages during the conflict resolution phase.

1. Introduction

Is requirements engineering (RE) a complicated addition to current knowledge engineering (KE)? Will the process of rationalising multiple conflicting viewpoints slow down the production of an expert system? If the costs of RE are prohibitive, then RE will rarely be applied.

In this paper, we present a simple RE strategy that is an extension to current KE techniques. Given assertions in rulebases (the A-boxes) from different stakeholders, we generate and critique a concept hierarchy (the T-box). Conflicts recognised in the T-box can be used to drive negotiation strategies amongst the different stakeholders. A general framework for this approach is described together with an instantiation using formal concept analysis (FCA) (Wille, 1982) and ripple-down rules (RDR) (Compton and Jansen, 1990).

This paper is organised as follows. Section 2 introduces RE and section 3 describes our RE framework that is instantiated in section 4. Related work and the conclusion are presented in Sections 5 and 6, respectively.

2. Requirements Engineering: A Review

Requirements engineering (RE) can be defined as "the elicitation and formulation of requirements to produce a specification " Easterbrook (1991, p.8). Current requirements engineering focuses on the maintenance of multiple concurrent viewpoints from different stakeholders (e.g. Easterbrook or Finklestein et al 1994).

There are a number of reasons why it is important to capture these multiple viewpoints rather than taking the approach that the different viewpoints must be captured into one specification. Tracking multiple perspectives is needed because:

1. Specification errors are often the cause of a poor choice between alternatives during the specification phase (Ramesh and Dhar 1992). By tracking multiple perspectives a history of the design rationale is provided so that when modifications are necessary they can be made more quickly (Easterbrook 1991) and based on the background that formulated them in the first place (Easterbrook 1991, Ramesh and Dhar 1992).
2. Many people are involved in projects requiring information to be passed between groups and phases. Individuals will forget and may change over time, subgroups have different roles and viewpoints (Easterbrook 1991).
3. Allows steps to be replayed and retraced. This has implications for reuse of specifications because if it is understood in what circumstances a certain path should be taken then these steps can be reapplied (Easterbrook 1991).
4. A more representative specification can be developed and a better framework for conflict resolution can be provided. The specification acts as both a contract and a communication channel (Easterbrook 1991).
5. Ownership is an important issue and by allowing multiple perspectives owned by the originator of that perspective we are more likely to motivate the user to participate in the resolution process. (Easterbrook 1991).

Our concept of a viewpoint corresponds to Finkelstein et al's (1989) formalisation of a viewpoint which includes: a style, area of concern, a specification, a work plan and a work record. This allows for an individual to hold a number of viewpoints and removes some of the problems of equating a particular viewpoint with an individual. Each owner of a viewpoint we call a stakeholder. When managing different viewpoints, conflict between different stakeholders must be handled.

3. Our Framework

Our viewpoint management framework has five steps shown in Figure 1. These five steps are iterative. For example, once a conflict has been detected and the decision is to modify an existing concept we go back to the first step of requirements acquisition to update the appropriate KBS.

4. An Implementation

Our general framework in Figure 1 has not committed to any particular implementation choices. We continue now looking at this framework but within the context of our instantiation. As a result a number of restrictions on the generality of our implementation are imposed.

Repeat

1. Requirements acquisition - Capture each viewpoint in a working knowledge based system (KBS). The KBS is an assertional knowledge base (A-box)[1], which we call the performance system, and a terminological knowledge base (T-box), which we call the explanation system, plus a set of cases. These cases can be divided into historical cases representing true observations from the domain; and hypothetical cases representing some desired functionality. Note that historical cases cannot be doubted while hypothetical cases can possibly be ignored.
2. Requirements integration - convert all KBS into a common format.
3. Concept generation - In this phase we add to the T-box for each individual KBS.
4. Concept comparison and conflict detection - Compare the T-boxes of each KBS and detect conflicts.
5. Negotiation - Employ a resolution strategy based on the type of conflict detected in step four. Output of this phase is fed back into phase one.

Until all stakeholders satisfied

Figure 1: A Framework for Requirements Engineering.

4.1 Phase One: Requirements Acquisition using a knowledge-based approach

We take a knowledge-based system (KBS) approach to requirements acquisition. In this paper, we focus on the case where the inputs are multiple A-boxes provided from multiple experts, the T-boxes are empty and the set of cases for each A-box is not empty, that is $A \neq \emptyset$, $T = \emptyset$ and $X \neq \emptyset$, where A, T and X denote an assertional KBS, terminological KBS and set of cases, respectively. We place a further restriction that the A-box must be convertible into a decision table. It has been shown (Colomb 1993) that any decision tree or propositional KBS may be converted into a decision table. Other work (Richards and Compton 1997) has shown how ripple-down rule (RDR) systems can easily be converted into decision tables. Conversion to a decision table is also suitable for production rule-based systems and has been applied to a number of CLIPS KBS. For our purposes we distinguish between RDR systems and propositional rulebases which we call “standard rules”.

This phase is also the maintenance phase for once one cycle is completed it is vital to ensure that the changes made to the explanation system (T-box) output from Phase Five are reflected in the appropriate individual and shared performance systems (A-boxes). This study proposes the use of multiple classification RDR (MCRDR) (Kang, Compton and Preston 1995) for Knowledge Acquisiton (KA) and knowledge representation (KR). We adopt RDR because maintenance in RDR is a simple task that can be performed by the user and does not suffer from the side-effect problem which occurs in typical rule-based systems (Soloway, Bachant and Jensen 1987). An

[1] Assertional KBS are made up of executable assertions (such as rules) that assert the relationships between terms. Terminological KB consist of terms structured into inheritance networks (Brachman 1979). Their main building blocks are concepts and roles and they reason by determination of subsumption between concepts (Nebel 1991).

additional benefit of RDR, as mentioned above, is that the rule pathways map directly into a decision table and do not need intermediate conclusions to be mapped to primitive conditions as many rule bases require.

The RDR approach to KA is to run a case and show the user the system-assigned conclusions. If the user agrees with the conclusions given then they process the next case. If they do not agree with a conclusion they take the option to reclassify the case. Reclassification involves specifying the correct conclusion and picking some features in the case that justify the new conclusion. These features form the conditions of the new rule. The new rule is added as an exception to the rule that gave the misclassification. The case that prompts a rule to be added is stored in association with the new rule. When a new rule is added, the rule must distinguish between the present case and all the stored cases that can reach that rule. To do this, the expert is required firstly to construct a rule which distinguishes between the new case and one of the stored cases. If other stored cases satisfy the rule, further conditions are required to be added to exclude a further case and so on until no stored cases satisfy the rule. Stopping rules, which prevent an incorrect conclusion by providing a null conclusion are added in the same way. Surprisingly the expert provides a sufficiently precise rule after two or three cases have been seen (Kang, Compton and Preston 1995). Multiple Classification (MCRDR) is defined as the triple <rule,C,S>, where C are the children/exception rules and S are the siblings. All siblings at the first level are evaluated and if true the list of children are evaluated until all children from true parents have been exhausted. The last true rule on each pathway forms the conclusion for the case. Figure 2 shows an example of an MCRDR.

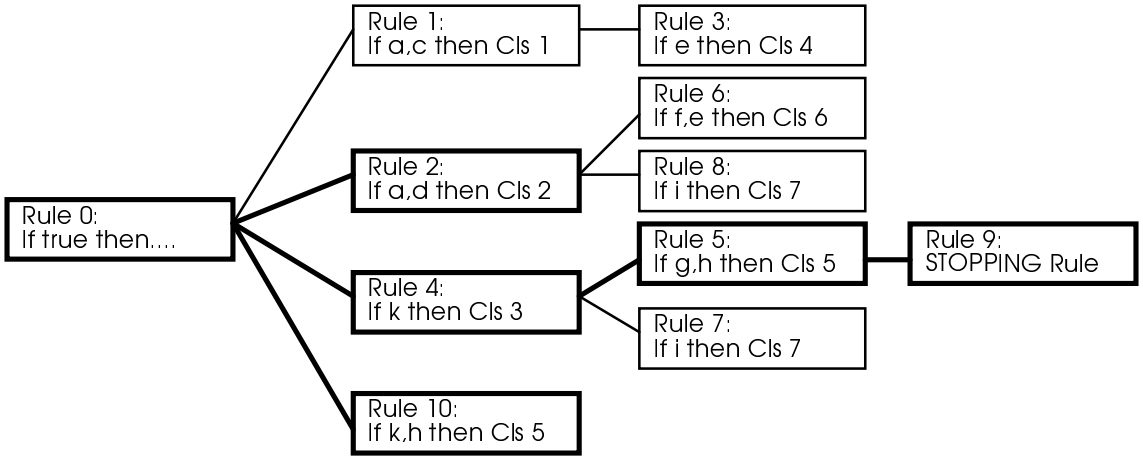

Figure 2. An MCRDR KBS.

The highlighted boxes represent rules that are satisfied for the case {a,d,g,h,k}. We can see that there are two independent conclusions for this case, Class 2 (Rule 2) and Class 5 (Rule 10). Rule 5 had been the cause of a conflict between viewpoints. To resolve this conflict it was decided that attribute g should be dropped. As described in Figure 6, the STOPPING RULE is used to say that this pathway should not fire, so even though the case satisfies Rule 5 that rule is stopped from being reported. We can see that Rule 10 now replaces the rule pathway for Rule 5 dropping the attribute g.

The greatest success for RDR has been the Pathology Expert Interpretative Reporting System (PEIRS) (Edwards et al 1993). PEIRS went into routine use with approximately 200 rules and grew in a four year period (1990-1994) to over 2000 rules. The system was maintained by the expert and the 2000 rules represents a development time of 100 hours.

As noted in our general framework, the input to Phase One also includes a set of cases. The importance of the set of cases is twofold. Firstly, on a general level, the cases are used in the negotiations as counterexamples for discussion. Seondly, using the RDR approach as described above, or other case-based technique, the cases are also used for initial KA and for modification of other views. The way this works is that when a concept is found to be in conflict, as one of our modification strategies outlined below (see Figure 6), we pass the case or cases associated with that concept to the other stakeholder for KA. This should either resolve the conflict or at least ensure that both parties have given their views given the same set of criteria. In Related Work we suggest how we can obtain cases.

4.2 Phase Two: Requirements integration

Requirements integration is the process of ensuring that all viewpoints are in formats that can be compared. Adopting our general framework, it may be that viewpoints have been captured using different KR. To avoid the requirement of mapping from all KR's used into all other KR's, that is N^2 mapping schemes, we convert all KRs into one format so that we only need 2N mapping schemes. As mentioned in Phase One, in our current implementation we can use any representation that maps into a decision table so that a common approach to subsequent phases can be taken. We explain in the next section why we have the decision table format restriction.

4.3 Phase Three: Concept Generation

In our general framework, an explanation system could already exist (that is, $T \neq \varnothing$ in Phase One). Alternatively, it could be supplemented or built in this phase. In the current work, we restrict ourselves to the case where $T = \varnothing$. The approach we have chosen is to begin with a performance system and later derive the explanation system. We start with a set of privately owned and defended A-boxes ($A_1..A_i$) written by some experts ($X_1..X_i$). The knowledge base also includes some data structures generated from previous cycles through Figure 1. These structures are:

- One circumvent table for each A-box. This table identifies which rules to skip in future RE sessions.
- One synonym table for the entire system. This table stores mappings of different terms to a common term.
- One delayIgnore table for the entire system. This table tags defines which T-box conflicts have been marked as "ignored" or "delayed" in the previous cycle.

For more on these tables, see Section 4.5.

We have taken the approach of starting with a performance system (A-box) and using that to derive an explanation system (T-box) because we see that defining and building models is inherently difficult and flawed. Easterbrook (1991) points out that one of the reasons why systems fail to meet the user's needs is that the original mental model of the user has not been captured in the final design model. We see the difficulty in capturing mental models as a contributing factor to the bottleneck associated with KA. We now show how we use FCA to make the leap from A-box to T-box.

A concept in FCA is comprised of a set of objects and the set of attributes associated with those objects. The set of objects forms the extent of the concept while the set of attributes forms the intent of the concept. Knowledge is seen as applying in a context and can be formally defined as a crosstable. We interpret the decision table in Figure 3 as a formal context where the rows are objects and the columns are attributes. An X indicates that a particular object has the corresponding attribute. This crosstable is used to find formal concepts.

	Source-borrower	source library	input book	input card	action check-in	action check-out	output book	output card	dest. borrower	dest. clerk
Borrower-check-in	X		X		X					X
Borrower-check-out	X			X		X	X	X	X	
Library-check-out		X	X			X	X	X	X	

Figure 3: Context of "Library from Borrower Viewpoint"

In Figure 3 we have the formal context "Library from the Borrower Viewpoint" with the set of objects = {Borrower check-in, Borrower check-out, Library check-out} and set of attributes = {source borrower, source library, input book, input card, action check-in, action check-out, output book, output card, destination borrower, destination clerk}. The crosses show where a relation between the object and attribute exists, thus the set of relations = {(Borrower check-in, source borrower), (Borrower check-in, input book),…,(Library check-out, destination borrower)}. Each row in the crosstable represents a concept. By finding the intersections of sets of attributes and the set of objects that share those attributes we are able to form new higher level abstractions. The set of concepts can be ordered using the subsumption relation ≤ on the set of all concepts which can be used to form a complete lattice. For a more detailed and formal treatment of our approach see Richards and Compton 1997.

In Figure 4 the concepts are shown as small circles and the sub/superconcept relations as lines. Each concept has various attributes and objects associated with it. The labelling has been reduced for clarity. All attributes of a concept are reached by ascending paths from the concept and all extents are reached by descending paths from the concept. The concept lattice provides "hierarchical conceptual clustering of the objects (via the extents) …. and a representation of all implications between the attributes (via its intents)" (Wille 1992, 497).

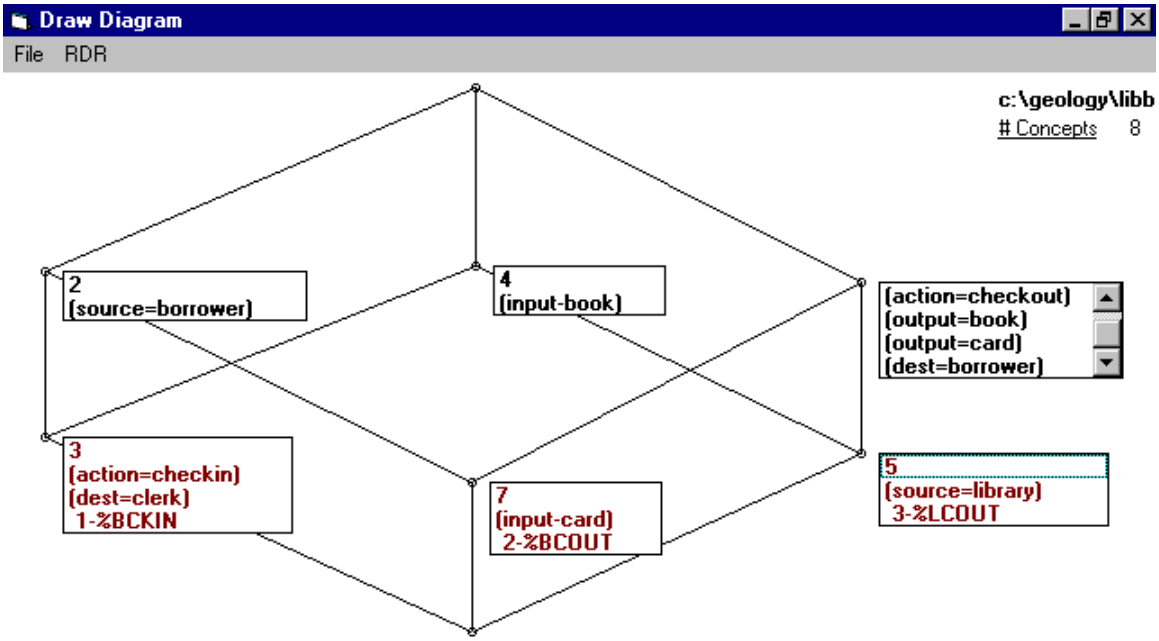

Figure 4:The Line Diagram for the Formal Context "Library from Borrower Viewpoint".
There are 8 concepts altogether. We can see that Concepts No. 5 and 7 share many attributes as they are both concerned with the check-out process. The differences between them are found in Concepts No. 2 and 4 which show that Concept No.5 covers the situation when the library inputs a book and Concept No. 7 concerns the situation where the borrower inputs a card.

4.4 Phase Four: Concept Comparison and Conflict Detection

A number of researchers offer different sets of conflict types (e.g. Easterbrook 1991 and Schwanke and Kaiser 1988). In this paper, we use the four quadrant model of comparison between experts developed Gaines and Shaw (1989). This model classifies two conceptual models as being in one of four states:

Consensus is the situation where experts describe the same concepts using the same terminology.
Correspondence occurs where experts describe the same concepts but use different terminology.
Conflict is where different concepts are being described but the same terms are used.
Contrast is where the there is no similarity between concepts or the terminology used.

In this paper we generally take a broader view of conflict to encompass inconsistencies that include the states of contrast, correspondence and conflict. Gaines and Shaw's model, however, does offer us greater precision in describing the nature of the conflict which is important in deciding how it can be handled. We more formally define the states of consensus and contrast according the FCA notion of a concept as a related set of attributes and objects. V denotes a View, C denotes a concept, A denotes a set of attributes and O denotes a set of objects. Figure 5 gives an example of each of the four states as they may appear on a concept lattice.

Consensus $\{V_1.C_i.A_j\} = \{V_2.C_k.A_L\}$ where $A_j = A_L$ and
$\{V_1.C_i.O_j\} = \{V_2.C_k.O_L\}$ where $O_j = O_L$

Contrast $\{V_1.C_i.A_j \cap V_2.C_k.A_L\} = \varnothing$ and
$\{V_1.C_i.O_j \cap V_2.C_k.O_L\} = \varnothing$

A concept not in a state of consensus (match found in another viewpoint) or contrast (completely different to all concepts in another viewpoint) is then either in a state of correspondence or conflict. The key to deciding which state it belongs to depends on the terminology. In our approach it would be up to the stakeholder to decide whether the terminology used for an attribute or object was the cause for two concepts not appearing at the same node. If more assistance for the user is desired, Gaines and Shaw (1989) have shown that the repertory grid technique can be used to identify where terminology is the cause of inconsistency.

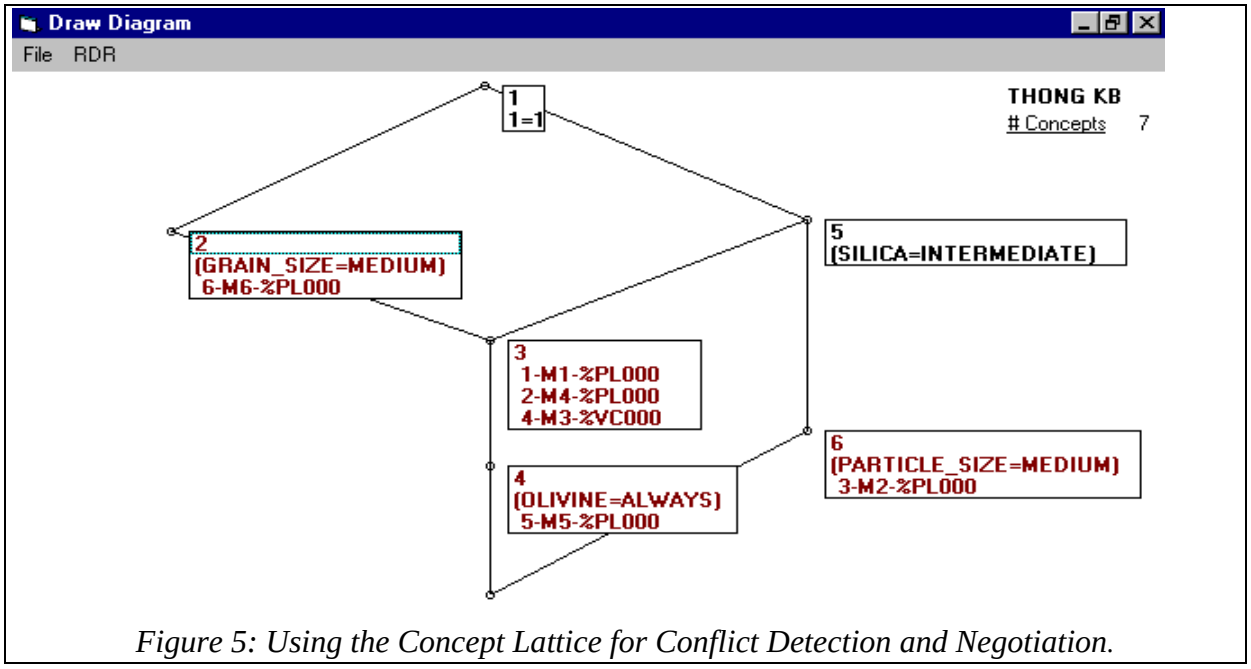

Figure 5: Using the Concept Lattice for Conflict Detection and Negotiation.

In Figure 5 we can see how the concept lattice can be used to identify the different types conflict. The line diagram, from our Windows implementation MCRDR/FCA, shows seven concepts generated from rules from six different viewpoints, shown as M1-M6. All views include the top Concept No. 1 which has the default condition. The diagram shows us that views M1 and M4 are in consensus. M3 is in conflict with M1 and M4 because although it shares the same set of attributes it has a different conclusion (%VC000 instead of %PL000). Concept No.2 shows that view M6 is a superset of M1, M4, M3 and M5, therefore in a state of partial conflict with those views, and is in a state of contrast with view M2. M2 is in a state of correspondence with M1, M4 and M5 because it shares the conclusion (%PL000), the attribute (SILICA = INTERMEDIATE) but uses (PARTICLE_SIZE = MEDIUM) instead of (GRAIN_SIZE = MEDIUM). If the term

PARTICLE were changed to GRAIN then M2 would also appear at Concept No 3. Concept No. 4 is in partial conflict with Concept No. 3 because it has the additional attribute (OLIVINE = ALWAYS). If it were decided during the next conflict negotiation phase that the attribute was not relevant it could be removed and then concept views M1, M4 and M5 would be in consensus. Let us now look further at the various resolution operators available.

4. 5 Phase Five: Conflict Negotiation

Before we can decide how to fix a detected inconsistency we need to provide a conflict resolution strategy. There are a number of resolution methods which include negotiation, arbitration, coercion and education (Strauss 1978). Negotiation is the most appropriate within the assumed context of parties of equal status and ability. As Easterbrook (1989) points out, a good solution will require creativity and creativity is not something that can be automated. However, since automation is a fundamental goal of requirements engineering research we extend our approach beyond a general, genial chat by offering as much automated assistance for this step as possible.

Each RE researcher appears to use a different set of resolution strategies (e.g. Easterbrook 1991, Thomas 1976). Easterbrook and Nuseibeh (1996) offer five categories that covers the actions we have found necessary. These are:

- Resolve, correct any errors;
- Ignore, no action is performed;
- Delay, identify the existence of the inconsistency but defer action until a later date;
- Circumvent, identify the existence of the inconsistency so it can be avoided;
- Ameliorate, reduce the degree of inconsistency. This action requires analysis and reasoning.

Resolving conflict will involve correcting all errors. If the cause of disagreement is differences in terminology, correspondence in the Gaines and Shaw four state model, then one technique is to up-date all views to conform to an agreed upon set of terminology. This option is probably not satisfactory to the various stakeholders and also means that the history of changes is being lost or altered. A simple and more appropriate solution is to use synonym tables which map terms from individual views into a shared terminology which are then used for comparison.

Another way in which conflict may be resolved is through the addition or deletion of attributes or objects. The conflict may be that the set of attributes or objects are partially shared by another concept. To bring these concepts into a state of consensus it may be decided to drop or add attributes or objects. As mentioned in Phase One, part of our automated support for negotiation is the ability to produce a case associated with the object (rule) that is in question. The cases associated with all objects that can be reached by downward paths are also relevant to the discussion. The closer, distance measured in number of objects separating the two nodes, the object is to the concept in question the more relevant the case should be considered. New attributes or objects could also be added by showing the associated case to the other party and using that case for KA. Alternatively, if a hypothetical case is shown to be impossible, then the

rules based on this case should be dropped. The decision of what action to take is made in this last phase and performed in Phase One. In Figures 6(a) and (b) we provide a summary of the strategies applicable to standard rules and MCRDR. Note that:

- In each strategy for handling attributes or objects from standard rules we have added the requirement that some sort of checking should be done to ensure that there are not unknown side effects elsewhere in the rulebase. With MCRDR we get this checking for free because we know that no previously correctly classified cases can become misclassified with the RDR approach to KA and exception structure.
- RDR has a very tight link between rules and cases. This link is not defined for standard rule bases. Hence, handling case addition/deletion is not defined in standard rule bases (see Figure 6(b)).

	Add Attribute	Delete Attribute	Add Object	Delete Object
Standard Rules	Add attribute to rule. Check effect on other rules.	Remove attribute from rule. Check effect on other rules.	Add new rule. Check effect on other rules.	Remove rule. Check effect on other rules.
MCRDR	Defined - use existing KA approach	Add stopping rule to rule in error. Add new rule at top level with the old rule minus the attribute to be removed.	Perform KA using the case associated with the concept which has the desired object.	Add a stopping rule.

Figure 6(a) The resolution strategy for handling attribute and objects.

	Adding Real Cases	Deleting Real Cases	Adding Hypo-thetical Cases	Deleting Hypothetical Cases
Standard Rules	N/A	N/A	N/A	N/A
MCRDR	Show to all	ILLEGAL	Show to all	Drop rules - check refinement rules if they should be dropped or stopped and a new rule, without the dropped rule conditions, added.

Figure 6(b) The resolution strategy for handling cases.

We plan to strengthen our negotiation strategies by offering filtering rules which guide the dialogue between the stakeholder and the system. One filter is the use of preference criteria to guide the stakeholders in deciding what part of the view should be considered first as a candidate for change. If we consider the concept lattice structure where objects belonging to a concept are reached by descending paths and attributes are reached by ascending paths then we can say that if an attribute at the bottom of a pathway or an object at the top of a pathway is to be removed then we know that no other concepts will be affected and this can be performed without further investigation. This strategy can be useful for example, if it had been agreed upon that two attributes were equivalent and one was at the bottom of a pathway and the other higher up, then it would be advisable to remove the lower attribute. Other templates or KA scripts (Gil and Tallis

1997) can be used to guide the user with revising their KBS and ensuring that each of the rules related to the change are modified and tested.

The last four resolution strategies are relevant for situations in which a complete resolution can not be negotiated and each one has its appropriate usage. For example, ignoring is a useful strategy where the issue is not that important or pursuing it is not worth the effort or harm it may cause to the end solution. These approaches can be termed as living with inconsistency or 'lazy' consistency (Narayanasway and Goldman 1992) and can be compared to fault-tolerant systems that continue to function after non-critical failures occur. It has been argued that enforcing removal of all inconsistency "constrains the specification unnecessarily" and "tends to restrict the development process and stifle novelty and invention" (Finkelstein et al 1994, p.2 & 4). They see that consistency is necessary within a viewpoint but partial consistency between viewpoints is allowable.

We also accept that living with inconsistency will be necessary and use tags to identify the status of the conflict. The use of tags is similar to the use of "pollution markers" (Balzer 1991) that act as a warning that code may be unstable or that the users should carefully check the output. Pollution markers can be used to screen inconsistent data from critical paths that must have completely consistent input. If it is the concept that is being ignored or delayed, we mark the concept in the shared T-box since there is not necessarily a one-to-one correspondence between rules and concepts. This updated T-box is used as input in the next T-box generation. However, we take a different approach to circumvention because the avoidance of certain unstable parts of the requirements is more relevant to the rules. Therefore, we tag a rule as "circumvented" in the individual A-boxes. When the new T-boxes are generated these rules will not be included.

The resolution strategies shown in Figures 6(a) and (b) are also applicable to the strategy of amelioration. However, the result is not consensus but a reduction in the extent of the conflict. Amelioration results in bringing concepts closer together. If we think in terms of the concept lattice we would be shortening the distance between the two concepts.

4.6 Evaluation and Discussion

To determine that our RE strategy is resolving conflict we need to employ some measures of the degree of conflict before and after. By computing a score for each concept in each viewpoint compared to each other viewpoint and taking the total of these scores we can check that the degree of conflict after the RE process is less than at the start. We assign of score of 0 to a concept found to be in a state of consensus with a concept in another viewpoint, since the distance between them is zero. For concepts in a state of conflict we take the number of attributes (conditions) that they have but do not share divided by the total number of attributes. This assumes that the two concepts share the same object (conclusion). If they do not then it appears that they are not meant to represent the same concept so that comparison is not meaningful. For concepts in a state of contrast (no partial or complete match in the other viewpoint) we assign a score of 1, which is the same result as if we used the conflict measure since the number of

attributes not shared divided by the number of attributes is equal to one. Concepts in a state of correspondence are treated the same as concepts in conflict since we are ignoring the reason for the differences and are just interested in the size of the difference. Once terminology differences are reconciled such concepts will move into one of the other states and be handled accordingly.

The framework described in this paper has been initially tested using the SISYPHUS III (Shadbolt 1996) geology domain. The results from that experiment are reported in a sequel paper to this one, submitted to the Knowledge Acquisition Workshop to be held in Banff in 1988. In the sequel paper, we have used the measures in this section to determine how well our RE strategy was working. By using the synonym table, the resolution operators for adding and deleting attributes and cases shown in Figure 6 and employing the various resolution tags *circumvent, delay* and *ignore* we were able to achieve more than a 50% reduction in the amount of conflict for the example given. We also note that the RE extension to the MCRDR/FCA tool that already existed was less than a 2% increase in the existing 5000 lines of code. We estimate that a full implementation would require no more than a 10% increase. The interested reader is directed to the sequel paper for a full description of how each of the five requirements engineering phases can be applied and evaluated.

5. Related Work

Starting with a performance system and deriving an explanation system, is in complete contrast to mainstream KA research where the focus is on building domain (Chandrasekaran and Johnson 1993, McDermott 1988, Puerta et al 1992, Schreiber, Weilinga and Breuker 1993 and Steels 1993) and/or ontological (Guha and Lenat 1990, Patil et al 1991, Pirlein and Studer 1994) models first and using these to develop a performance system. As explained earlier, models are difficult to capture and unreliable and we prefer to capture knowledge the peformance knowledge that can be demonstrated and observed. RDR has given us a reliable method for capturing and maintaining performance knowledge and FCA is the mechanism that lets us derive the explanation system.

In our framework, cases play a critical role and we have assumed that cases are available. One viable option for the purposes of RE is *use cases* since they satisfy our need for sets of attributes and outcomes and are “primarily an approach to discovering requirements from a user-centred viewpoint” (Rumbaugh 1994, p.23). They could be used in conjunction with RDR or as direct input into the decision table and maintenance would consist of modification to the use cases. So that we don’t forget parts of the system we first enumerate the actors, which are external agents that require services from the system, and then the use cases. Specific values, not generalisations should be plugged into the cases so that thinking is grounded in precise examples. Rumbaugh suggests first building a system which contains the domain model and then the application model using use cases. We can equate the domain model and application model to our T-box and A-box, respectively. What we are advocating is to use the cases to build the A-box, the rules, from which we derive the T-box, the concept hierarchy.

6. Conclusion

We have argued here for a different view of RE. Standard RE is an early-software lifecycle issue. The viewpoint resolution technique discussed here can be performed whenever we have some A-box (rules) and cases. Initially, cases will be hypothetical and the rules sets small (snippets of known business processes). Finally, cases will be "live" data and rule sets will be large. In either case we can apply our technique. That is, our "RE Tool" can be applied right throughout the system development life cycle.

We have described an RE framework as an extension to an existing KE technique. RDR and FCA were used as subroutines within our RE system. FCA was used to build explanatory T-boxes from performance A-boxes. This approach is applicable to any representation which can be mapped into a decision table. However, during our discussion on resolution strategies, we noted that certain representations offered advantages. For example, when adding an attribute in a standard propositional rule base, the effects of this addition had to be checked all over the KB. Such a check comes for free in RDR.

This approach has also addressed a drawback of standard RDR. RDR systems have been shown to be useful for single expert knowledge acquisition. In such a situation, RDR offers a good performance module, but a poor explanation module. However, in the case of multiple experts, an explanation system is required since experts must trade off their competing views. FCA allows us to build an explanatory T-Box from an A-box initialised by RDR.

References

Balzer, R. (1991) Tolerating Inconsistency *Proceedings of 13th International Conference on Software Engineering (ICSE-13)* Austin, Texas, USA, 13-17th May 1991, 158-165; IEEE Computer Society Press.

Brachman, R.J. (1979) On the Epistemological Status of Semantic Networks In Findler, N.V. (ed) *Associative Networks: Representation and Use of Knowledge by Computers* Academic Press-50.

Chandrasekaran, B. and Johnson, T. (1993) Generic Tasks and Task Structures In David, J.M., Krivine, J.-P. and Simmons, R., editors *Second Generation Expert Systems* pp: 232-272. Springer, Berlin.

Colomb, Robert.M. (1993) Decision Tables, Decision Trees and Cases: Propositional Knowledge-Based Systems *Technical Report No. 266* Key Centre for Software Technology, Department of Compter Science, The University of Queensland, Australia.

Compton, P. and Jansen, R., (1990) A Philosophical Basis for Knowledge Acquisition. *Knowledge Acquisition* 2:241-257

Easterbrook, Steve (1991) Elicitation of Requirements from Multiple Perspectives *PhD Thesis*, Department of Computing, Imperial College of Science, Technology and Medicine, University of London, London SW7 2BZ.

Easterbrook, S. and Nuseibeh, B. (1996) Using Viewpoints for Inconsistency Management *BCSEEE Software Engineering Journal* January 1996:31-43.

Edwards, G., Compton, P., Malor, R, Srinivasan, A. and Lazarus, L. (1993) PEIRS: a Pathologist Maintained Expert System for the Interpretation of Chemical Pathology Reports *Pathology 25*: 27-34.

Finkelstein, A.C.W., Goedicke, M., Kramer, J. and Niskier, C. (1989) Viewpoint Oriented Software Development: Methods and Viewpoints in Requirements Engineering In *Proceedings of the Second Meteor Workshop on Methods for Formal Specification* Springer Verlag, LNCS.

Finkelstein, A., Gabbay, D., Hunter, A., Kramer, J. and Nuseibeh, B. (1994) Inconsistency Handling in Multi-Perspective Specifications *IEEE Transactions on Software Engineering* 20(8):569-578.

Gaines, B. R. and Shaw, M.L.G. (1989) Comparing the Conceptual Systems of Experts *The 11th International Joint Conference on Artificial Intelligence* :633-638.

Gil, Yolanda and Tallis, Marcelo (1997) A Script-Based Approach to Modifying Knowledge Bases In *Proceedings of the Fourteenth National Conference on Artificial Intelligence and Ninth Innovative Application of Artificial Intelligence Conference* AAAI Press/ MIT Press, Cambridge, Massachusetts.

Guha, T.V., and Lenat, D.B. (1990) CYC:A Mid-Term Report *AI Magazine* 11(3):32-59

Kang, B., Compton, P. and Preston, P (1995) Multiple Classification Ripple Down Rules: Evaluation and Possibilities *Proceedings 9th Banff Knowledge Acquisition for Knowledge Based Systems Workshop* Banff. Feb 26 - March 3 1995, Vol 1: 17.1-17.20.

McDermott, J. (1988) Preliminary Steps Toward a Taxonomy of Problem-Solving *Methods Automating Knowledge Acquisition for Expert Systems* Marcus, S (ed.) Kluwer Academic Publishers, pp: 225-256.

Narayanaswamy, K. and Goldman, N. (1992) "Lazy Consistency": A Basis for Cooperative Software Development *Proceedings of International Conference on Computer-Supported Cooperative Work (CSCW'92)* Toronto, Ontario, Canada, 31 October- 4 November, 257-264; ACM SIGCHI & SIGOIS.

Nebel, B. (1991) Terminological Cycles: Semantics and Computational Properties In John Sowa (ed) *Principles of Semantic Networks: Explorations in the Representation of Knowledge* Morgan Kaufmann Publishers, Inc. California, 331-361.

Patil, R. S., Fikes, R. E., Patel-Schneider, P. F., McKay, D., Finin, T., Gruber, T. R. and Neches, R., (1992) The DARPA Knowledge Sharing Effort: Progress Report In C. Rich, B. Nebel and Swartout, W., *Principles of Knowledge Representation and Reasoning: Proceedings of the Third Int. Conference* Cambridge, MA, Morgan Kaufman.

Pirlein, T and Struder, R., (1994) KARO: An Integrated Environment for Reusing Ontologies *European Knowledge Acquisition Workshop '94*, Springer Verlag

Puerta, A. R, Egar, J.W., Tu, S.W. and Musen, M.A. (1992) A Mulitple Method Knowledge Acquisition Shell for Automatic Generation of Knowledge Acquisition Tools *Knowledge Acquisition 4*(2).

Ramesh, B. and Dhar, V. (1992) Supporting Systems Development by Capturing Deliberations During Requirments Engineering *IEEE Transactions on Software Engineering* 18(6):498-510.

Richards, D. and Compton, P. (1997c) Uncovering the Conceptual Models in Ripple Down Rules In Dickson Lukose, Harry Delugach, Marry Keeler, Leroy Searle, and John F. Sowa, (Eds) (1997), *Conceptual Structures: Fulfilling Peirce's Dream, Proceedings of the Fifth International Conference on Conceptual Structures (ICCS'97)*, August 3 - 8, University of Washington, Seattle, USA, Lecture Notes in Artificial Intelligence, Springer-Verlag, Number 1257, Berlin pp:198-212

Rumbaugh, James (1994) Getting Started: Using Use Cases to Capture Requirements *JOOP* September 1994:8-23.

Schreiber, G., Weilinga, B. and Breuker (eds) (1993) KADS: A Principles Approach to Knowledge-Based System Development *Knowledge-Based Systems* London, England, Academic Press.

Schwanke, R.W. and Kaiser, G.E. (1988) Living with Inconsistency in Large Systems *Proceedings of the International Workshop on Software Version and Configuration Control* Grassau, Germany, 27-29 January 1988, 98-118;B.G. Teubner, Stuttgart.

Shadbolt, N., (1996)URL:http://www.psyc.nott.ac.uk/aigr/research/ka/SisIII

Soloway, E, Bachant, J. and Jensen, K. (1987) Assessing the Maintainability of XCON-in-RIME: Coping with Problems of a very Large Rule Base *Proceedings of the Sixth International Conference on Artificial Intelligence* Vol 2:824-829, Seattle, WA: Morgan Kaufman.

Steels, L. (1993) The Componential Framework and Its Role in Reusability In David, J.M., Krivine, J.-P. and Simmons, R., editors *Second Generation Expert Systems* pp: 273-298. Springer, Berlin.

Strauss, A. (1978) *Negotiation: Varieties, Contexts, Processes and Social Order* Jossey-Bass Publishers, San Francisco, CA.

Thomas, K. (1976) Conflict and Conflict Management In Duneette (ed) *Handbook of Industrial and Organisational Psychology* Rand McNally College Publishing Co.

Wille, R. (1982) Restructuring Lattice Theory: An Approach Based on Hierarchies of Concepts In *Ordered Sets* (Ed. Rival) pp:445-470, Reidel, Dordrecht, Boston.

Wille, R. (1992) Concept Lattices and Conceptual Knowledge *Systems Computers Math. Applic. (23***)** 6-9:493-515.