CHAPTER 12: RESULTS OF THE FASTFIX TRIAL

12.1 Chapter Outline

This chapter presents the results of the FastFIX prototype software trial undertaken at HTG during November 2005. The software trial showed that the 7Cs system and FastFIX prototype was able to support multiple users in collaboratively building the knowledge base using both a top-down rule-driven and bottom-up case-driven manner, with minimal training and supervision. Users had no problems working from a top-down rule-driven perspective, as well as a bottom-up case-driven perspective and the system did not appear to demand any significant additional knowledge engineering load than could be expected from a purely bottom-up case-driven RDR approach. As well, it was demonstrated that separately tracking the live versus registered RuleNodes captured the frequently occurring Case drop-through events in the rapidly evolving knowledge base, and thereby supported and maintained the integrity of the knowledge base. The tracking of live versus registered case-RuleNode associations facilitated the conflict resolution required for collaborative knowledge acquisition.

12.2 User Activity

The following table summarises user interaction with the FastFIX prototype during the software trial. 12 users registered themselves within the system, including the author (User ID 12). Most of the registered users were onlookers – managers, team leaders and other interested parties. There were three main contributors from the HTG side with user IDs 1, 3 and 6. These contributors were able to use the system with minimal training and supervision (less than 60 minutes per contributor). Contributors commenced by providing trouble-shooting knowledge for the most frequently occurring dial-home errors: those with an XX2F error code that related to disk-drive errors.

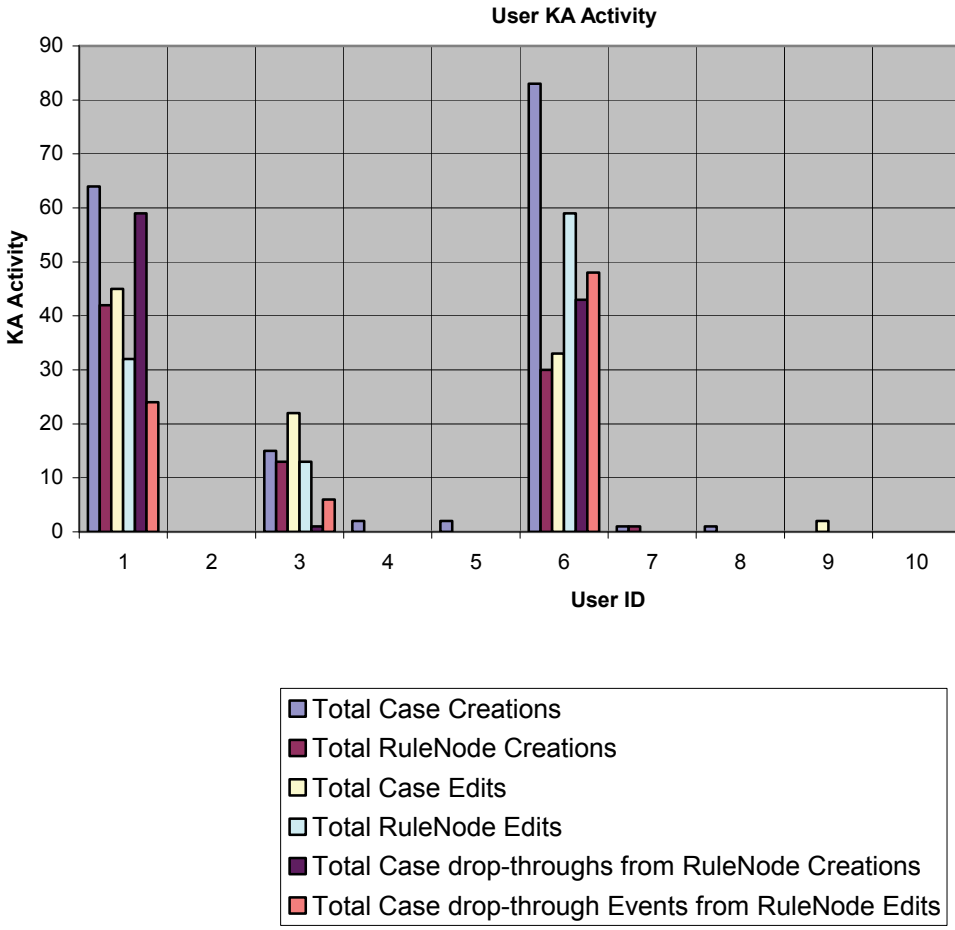
Table 21: User Activity in the prototype FastFIX system

User ID	1	2	3	4	5	6	7	8	9	10	11	12	Totals
Total Case Creations	64	0	15	2	2	83	1	1	0	0	1	3	**172**
Total RuleNode Creations	42	0	13	0	0	30	1	0	0	0	0	21	**107**
Total Case Edits	45	0	22	0	0	33	0	0	2	0	7	30	**139**
Total RuleNode Edits	32	0	13	0	0	59	0	0	0	0	0	37	**141**
Total Case drop-throughs resulting from RuleNode Creations	59	0	1	0	0	43	0	0	0	0	0	1	**104**
Total Case drop-through Events resulting from RuleNodeEdits	24	0	6	0	0	48	0	0	0	0	0	18	**96**

In total 172 cases and 107 RuleNodes were created. The total number of case edits was 139 and the total number of RuleNode edits was 141 demonstrating both the desire and capacity of users to contribute to knowledge evolution in this way. In total there were 104 case drop-throughs resulting from RuleNode creations. As well, there were 96 case drop-through events resulting from RuleNode edits where each of these events may have affected 1 or more cases (further details of the case drop-through events as a result of RuleNode Edits were not recorded during the trial).

The following figure provides a graphical representation of the data in Table 21.

Figure 42: User Activity in the prototype FastFIX system



12.3 Other Statistics

14 attributes were created in the system. 5 attributes were created prior to the software trial by the author, and 9 were created by users that were active in the system. These last 9 attributes were created by users in order to fetch more information in specific contexts that could be used to narrow down the classification of incoming cases.

Appendix N (page 440) shows the rule statement, classification and conclusion details of the accumulated RuleNodes. Note that the use of functions in the rule conditions at RuleNodes reduced the need for additional attributes to characterise (and hence separate) cases in the domain. Users took just 1-2 mins to add a rule, but it took 2-3 person weeks for one of the domain experts and the software engineer (the author) to define and develop the locate() function (described previously in Table 15, page 188) which was found to be necessary to sufficiently differentiate the incoming cases.

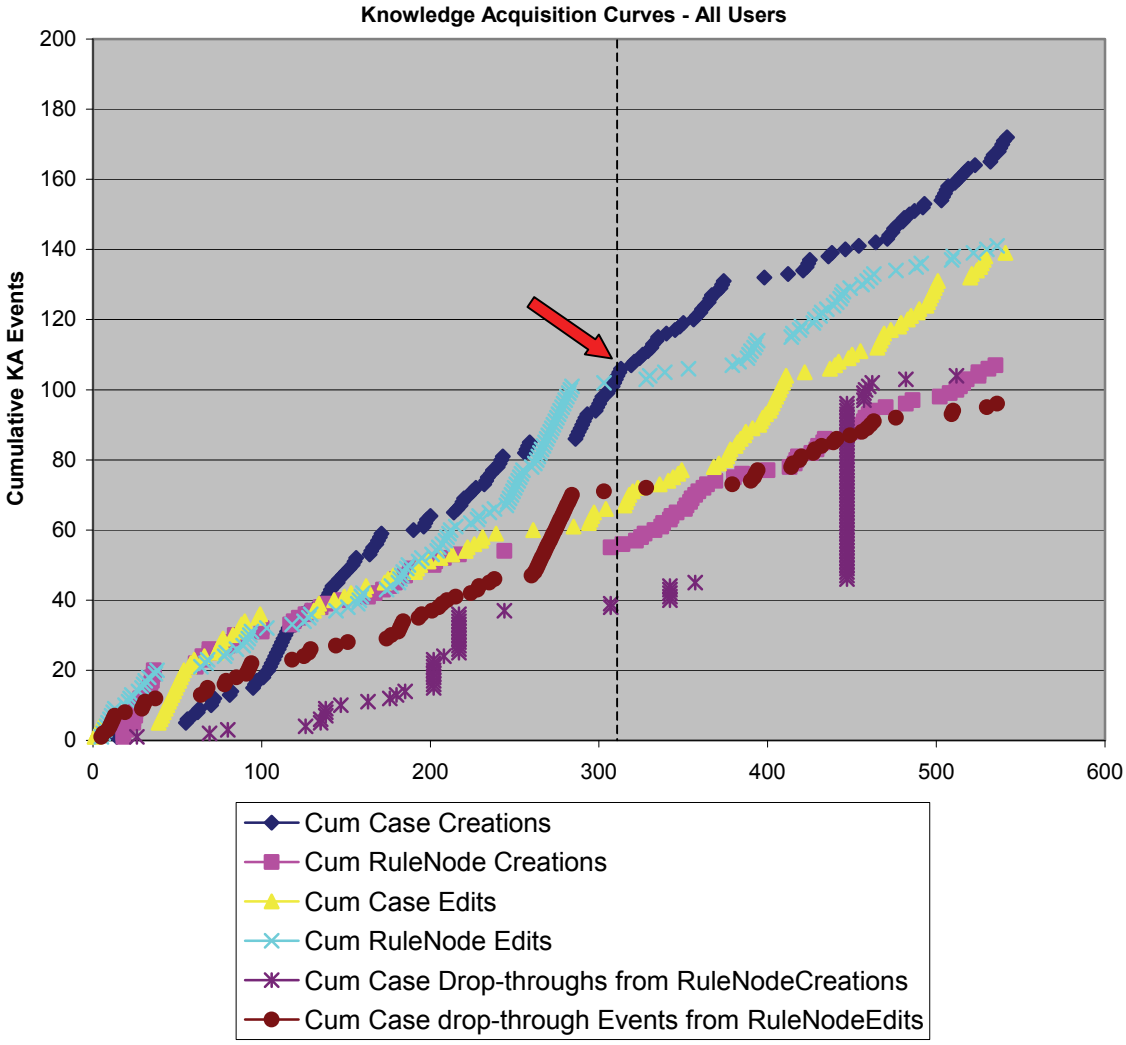
The maximum width of the rule tree was 54 RuleNodes. That width occurred directly beneath the root RuleNode. The maximum depth of the rule tree was just 4 nodes, including the root RuleNode.

Few people worked on the same case but numerous people worked on the same RuleNode. Changes to the same RuleNode occurred over a period of weeks. The separation between *live* and *registered* RuleNodes allowed users to be notified of the relevant knowledge evolution affects and their impact.

12.4 Combined KA Curves for All Users

The following figure shows the cumulative number of Case Creations, RuleNode Creations, Case Edits, RuleNode Edits, Case Drop-throughs resulting from RuleNode Creations, and Case Drop-through Events resulting from RuleNode Edits as knowledge was acquired by the system. The change histories for cases and RuleNodes recorded the timestamp at which each event occurred. In the following graph, a unique KA Event ID has been assigned to each unique timestamp recorded by the system. The KA Event ID was used to construct the x-axis. The vertical sets of case drop-throughs show that at times, multiple cases were affected by some of the RuleNode creations and/or RuleNode edits.

Figure 43:Cumulative Knowledge Acquisition Curves – All Users



12.5 Effectiveness of the Solution

After 7 person hours of effort in total the test team had captured 105 cases and 55 RuleNodes. The red arrow in each of following figures has been used to indicate this point in time. At this point the team had provided enough RuleNodes to automatically solve approximately 90% of disk drive (XX2F) related errors on errant equipment. These errors contribute to 30% of all dial-home errors seen by the system which account for 20% of the ~5,000 problem cases per day seen by the global ICT support centre. Hence after 7 hours of effort enough knowledge had been acquired to automatically provide solutions to more than 270 cases per day, without requiring the trouble-shooters to figure out the class of problem on hand, where to search for a solution, or what to search for.

Appendix N (page 440) shows the rule statement, classification and conclusion details of the accumulated RuleNodes. From Appendix N, it can be seen that significant cognitive processing and recall is required by experts to characterise the incoming problems and to know where-to-search and what-to-search-for to arrive at a relevant solution.

Say that each dial-home problem takes on average 15 minutes to solve, and that 1 minute of this time is spent in determining the problem and finding its solution. This represents a time saving of 1 mins * 270 cases per day, or 4.5 hours per day. Actually, the average solution search time is possibly a lot longer. One of the problems with manually searching for solutions is that if you haven't found the answer, you don't know if its just because your not searching for it correctly, or if its because a solution does not exist. The FastFIX system has the advantage that it unambiguously associates relevant solutions with their incoming problem classes. If the answer is unknown, FastFIX can provide that information (see Appendix N, RuleNode 49).

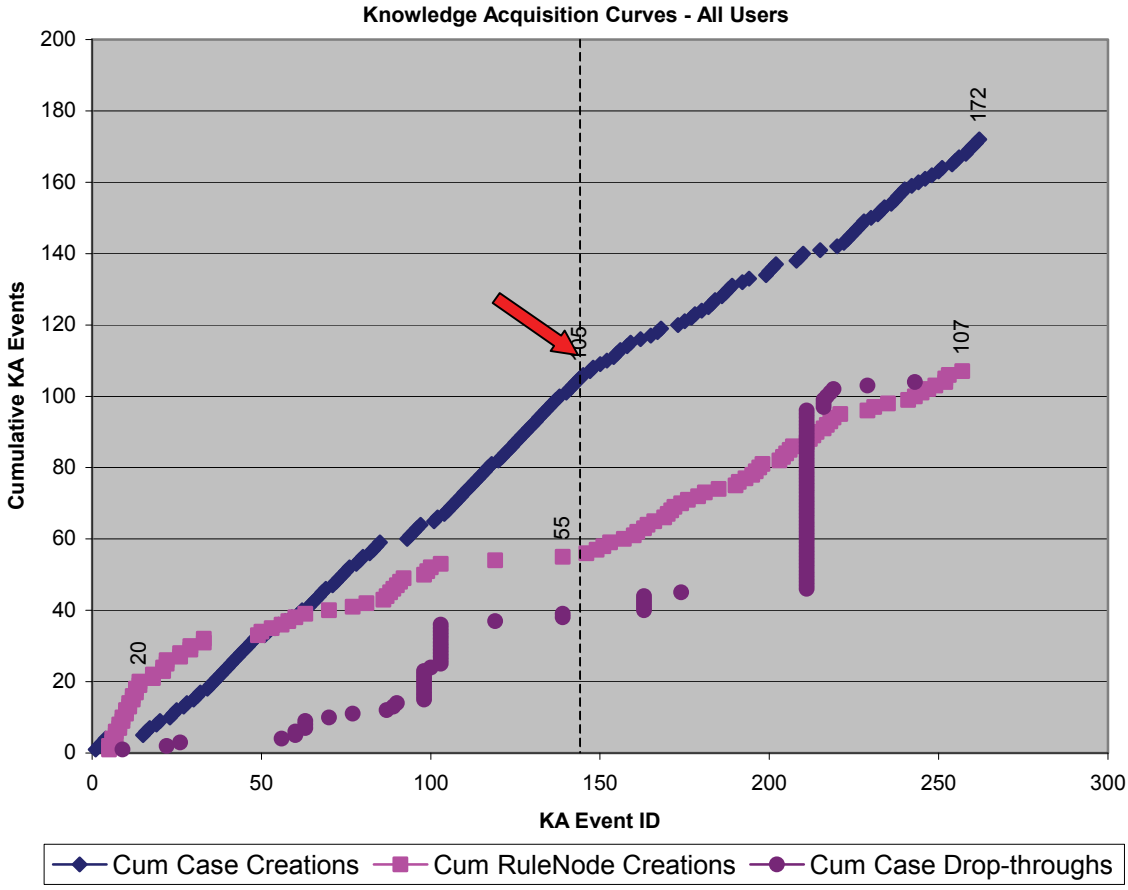
After the first 105 cases and 55 RuleNodes, the test team broadened the knowledge domain being covered to include non-drive queue problems. The trial of the prototype ceased after 107 RuleNodes and 172 cases had been accumulated. At that point, no new information was being gathered and it was time to look to the future at what additional features might enhance the system.

Note that these results parallel the results of the much longer PEIRS SCRDR software trial in which additional domains were incorporated incrementally after the pathologists had gained confidence in the performance of the system with the initially selected thyroid domain (Edwards, 1996, p90).

12.6 Case Drop-throughs

The following curve shows the cumulative case and RuleNode creations and Case drop-throughs resulting from RuleNode Creations in greater detail. A unique KA Event ID has been assigned to each unique timestamp captured in this subset of data and it has been used to construct the x-axis.

Figure 44: Cumulative Case and RuleNode Creation Curves



It can be seen from the data and observed in the above figure that the first 20 RuleNodes were provided to the system in a top-down (and hence rule-driven linear) manner. In contrast, RuleNodes 21 to 55 were provided mostly on the basis of cases seen in a case-driven bottom-up monotonically increasing and stochastic manner as described in Chapter 7. After this point, users were selective in choosing which cases to train the system with, choosing cases that were expected to be novel[183]. Hence RuleNodes 56 to 107 were provided to the system in a more top-down (and hence rule-driven linear) manner as for the first 20 RuleNodes.

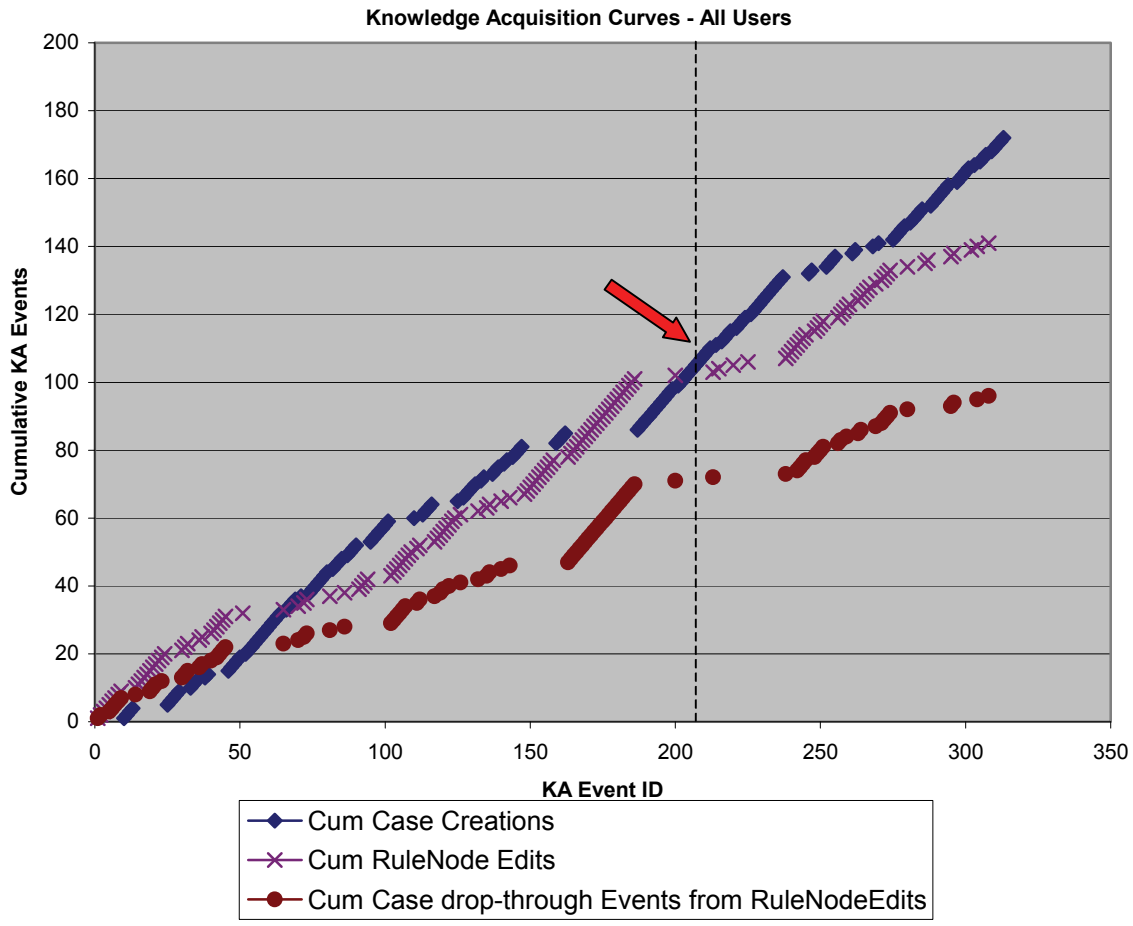
It is difficult to say how the ability of users to edit RuleNodes affects the overall case and RuleNode creation trajectories. If most of the RuleNode edits were cosmetic e.g. as a result fixing spelling mistakes then it can be expected that these KA trajectories would be little

[183] Users can do a similar thing to "active learning" in which the learner selects instances that it believes most uncertain.

affected by the RuleNode edits. However, if RuleNode editing represents a significant KA activity whereby genuinely new knowledge is being acquired, rather than existing knowledge being cosmetically corrected, then those RuleNode edit events should be added into the above case-driven KA trajectory. However, it was beyond the scope of this trial to examine this in any detail.

The following figure shows the Cumulative Case Creation and RuleNode Edit Curves.

Figure 45: Cumulative Case Creation and RuleNode Edit Curves



The number of RuleNode edits appears to grow in proportion to the number of cases seen by the system, which (according to the findings of Chapter 7) indicates that RuleNode editing tends to be a top-down knowledge acquisition activity.

12.7 Features Employed by Users

Users took advantage of the ability to label the classifications represented by RuleNodes as distinct from the conclusions at that RuleNode. 22 of 106 manually constructed RuleNodes had their classifications labelled (see Appendix N, page 440). As well, users also found cause to refer to conclusions at other RuleNodes (see Appendix N, RuleNode 76), although this feature was only introduced towards the end of the software trial.

The ability to combine text and hyperlinks in the conclusions at RuleNodes meant that multiple conclusions could be referred to at a single RuleNode (see Appendix N, RuleNode 62).

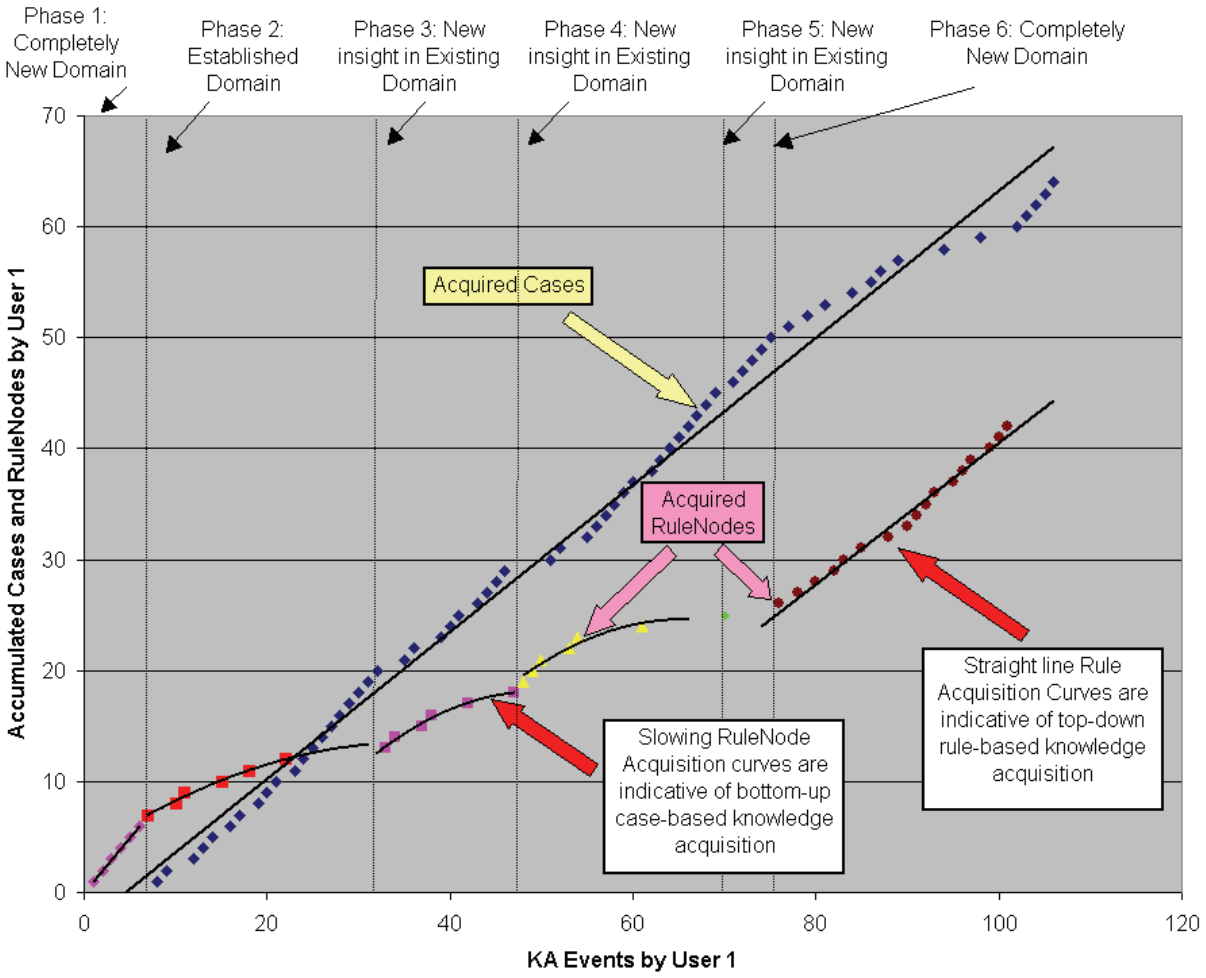
Users were able to add new attributes to the system and the system provided a way for users to effectively "work-up" their problem cases. This feature will need more testing and development in the future in order to improve its ease of use.

Users disabled 10 RuleNodes by editing them and creating rule statements that were `FALSE` (This feature was discussed in section 11.3.4 on page 190). In addition, users created 3 stopping RuleNodes to negate the validity of the parent RuleNode under certain rule conditions (refer to RuleNodes 29, 104 and 105 in Appendix N, page 440).

12.8 Experimental results

Figure 46 shows the KA trajectories for User 1.

Figure 46: KA Trajectory for a Human Expert.



In Figure 46 we see clear examples of both top-down rule-based and bottom-up case-based KA. In the figure both case and rule acquisitions are plotted on the x-axis as KA events so that the two types of KA trajectories can be shown together. User 1 shared 64 cases with the KBS and constructed 42 rules. The phases shown have been interpreted from the data.

In Phase 1, the user entered a completely new knowledge domain and began adding rules to the KBS in a top-down fashion by directly editing the condition mesh. At the end of this phase ground rules and background knowledge were established in the domain. A linear rule-driven KA trajectory was observed in this phase.

In Phase 2, the user began examining randomised repetitive incoming cases in the established domain, and the user augmented the condition mesh on an as-needs basis in the context of the

current case under review. A bottom-up case-driven KA trajectory was observed in this phase, as well as in phases 3, 4, and 5.

In Phase 3, the user continued to review cases and add rules to the condition mesh whenever the existing classes did not fit the case under review. However, in this phase, which occurred on a different day we see that the user may have gained new insights as to the differences between cases and the types of rules that may be added to the system. Phases 4 and 5 may represent further small quantum leaps in understanding and hence knowledge transfer ability – the user may be incrementally getting better at training the KBS, and / or the KBS is incrementally getting better at receiving the new knowledge.

In Phase 6 the user entered a completely new knowledge domain. Again we see that the user has chosen to populate this new domain in a top-down fashion by directly editing the condition mesh and adding new rules independent of any cases seen. The KA trajectory is therefore linear during this phase.

As shown in the next section, the KA curves for each of main contributors to the FastFIX KBS also demonstrated the transition between rule-based and case-based KA trajectories. It is very difficult to obtain statistically significant results for human driven KA experiments due to the natural variability of the knowledge transfer coefficients (such as those discussed in Chapter 7 on page 92 and Appendix Q on page 473). However, as sanity check the observed data aligns well with the predictions offered by the case-driven KA model proposed in Chapter 7.

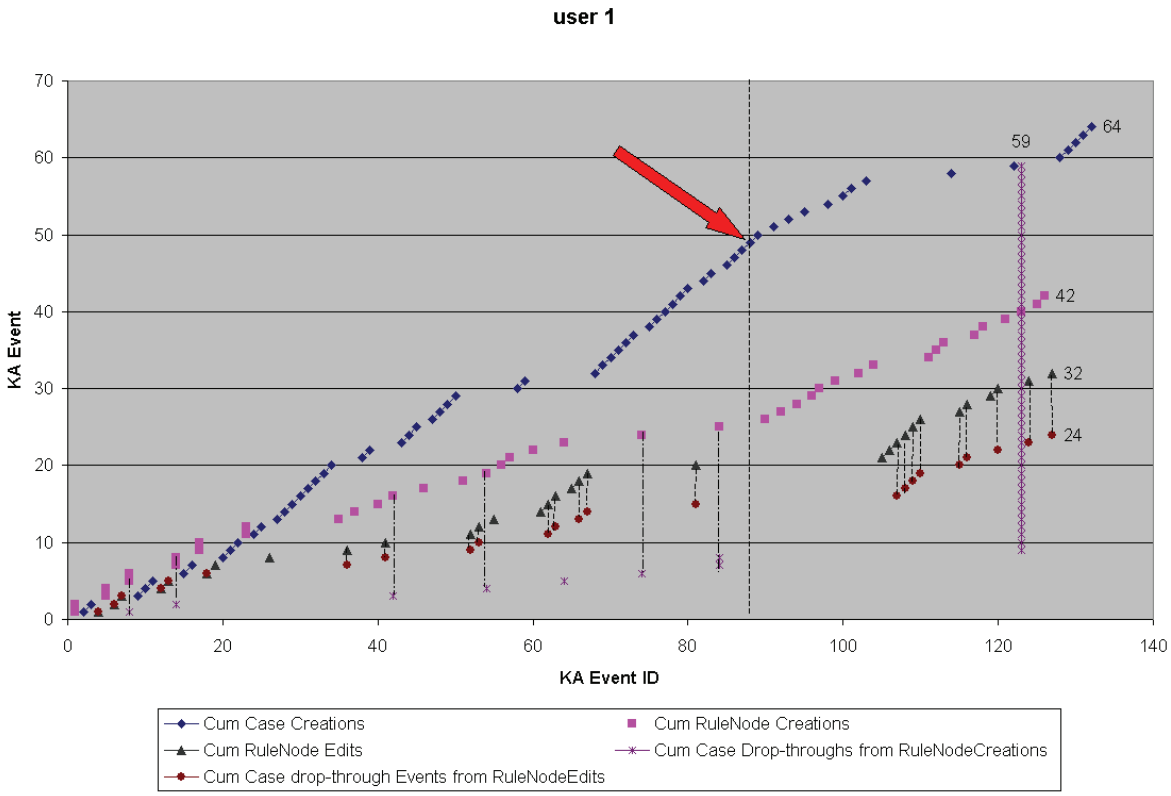
12.9 Individual KA Curves

Individual KA Curves are displayed in the next four figures for the most active users (with User IDs 1, 3, 6, and 12) in the system, including the author (User ID 12). Vertical lines have been included in the graphs to show the co-occurrence of RuleNode Creations and their resultant case drop-throughs, as well as RuleNode Edits and their resultant case drop-through events. Case edit events have been left out of the curves to allow the rate of RuleNode accumulation to be compared with the rate of Case accumulation.

12.9.1 User 1

From Figure 46 and Figure 47 it appears that RuleNodes are acquired by User 1 bottom-up prior to the domain change, and top-down thereafter. Figure 47 shows the case drop-through events resulting from edits to existing RuleNodes, as well as RuleNode creations by this user.

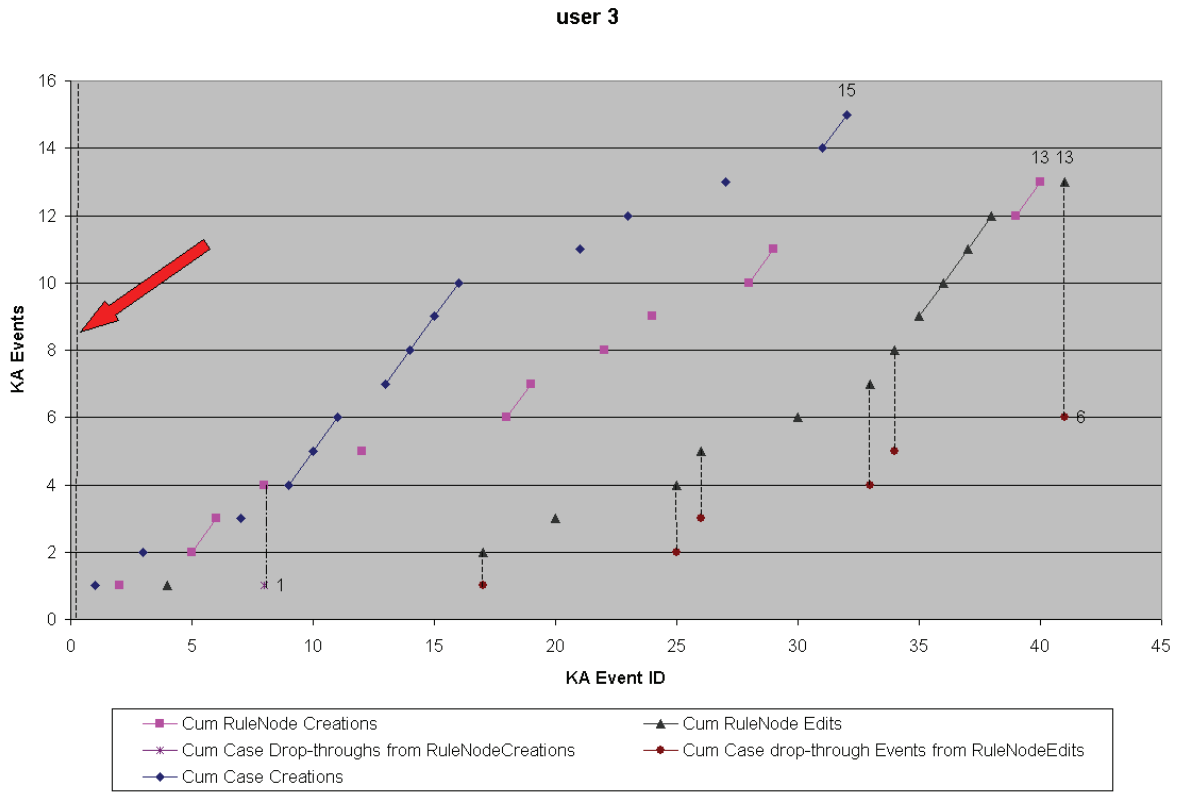
Figure 47: User 1



12.9.2 User 3

In the following figure, User 3's KA activity only started once the domain had been expanded to cover the non drive-queue problems. There appears to be a steady rate of accumulation of both cases and RuleNodes, although not much data is available for User 3.

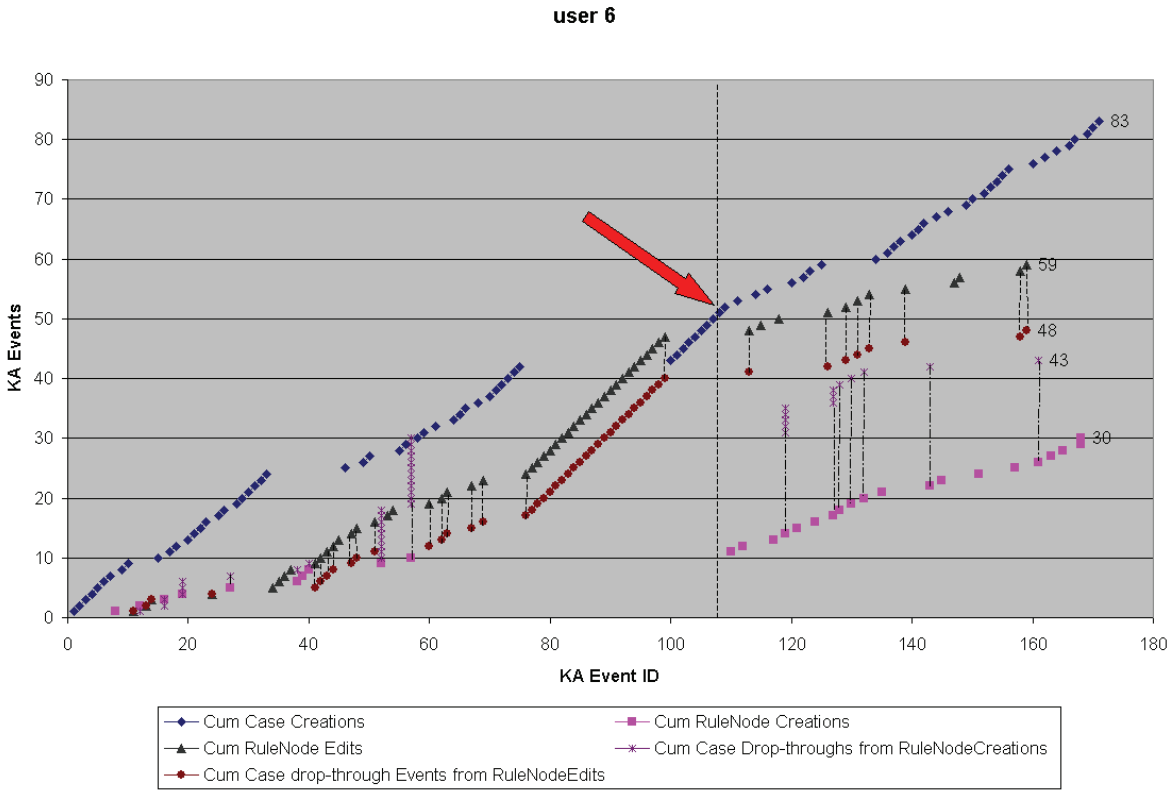
Figure 48: User 3



12.9.3 User 6

In the following figure, User 6's KA curves show a steady rate of accumulation of both cases and RuleNodes. As for User 1 it appears that RuleNodes are acquired bottom-up prior to the domain change, and top-down thereafter. User 6 undertook a major RuleNode editing activity between KA event 80 and 100. This effort was focussed on increasing the scope of the rule statements in a number of RuleNodes to not just consider 0F.XX2F errors, but to also cater for XX.XX2F errors.

Figure 49: User 6

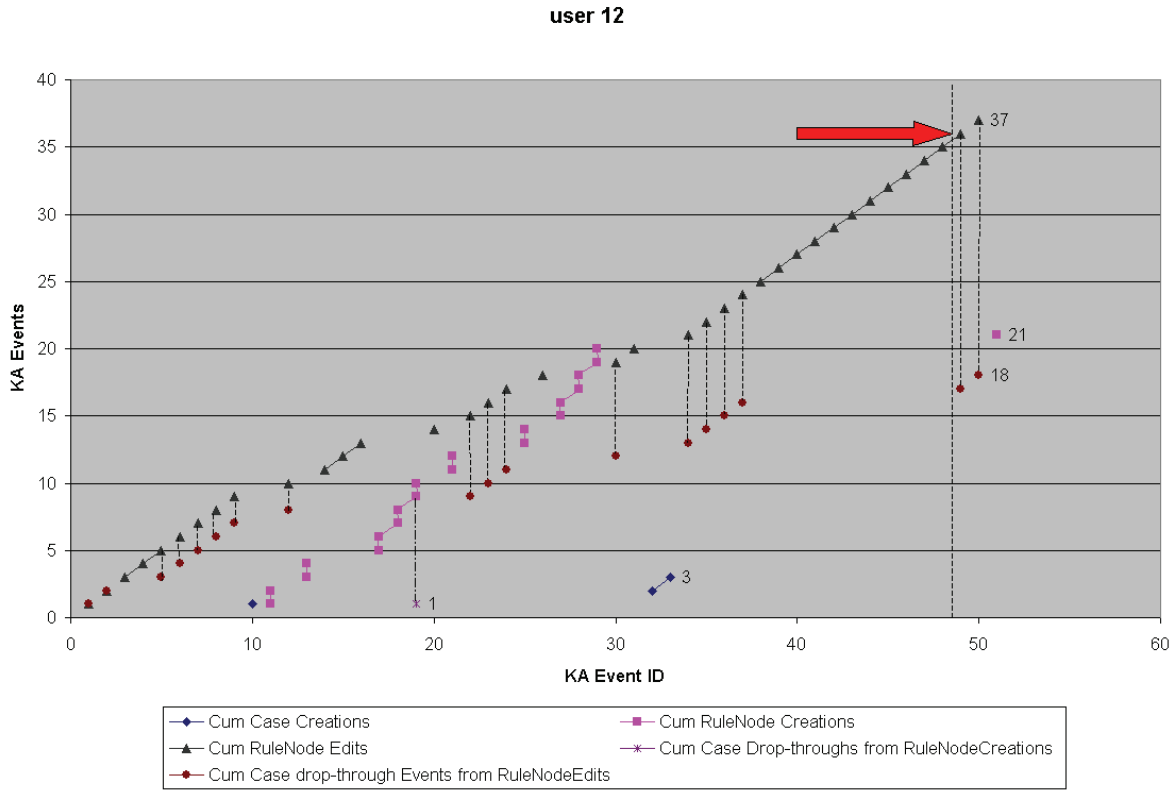


12.9.4 User 12

In the following figure, User 12's (i.e. the author and researcher's) KA curves show a focus on RuleNode edits in the early phases. At this point the system was still under development so both the users and the system were changing in the way they interacted with each other. User 12 created the first 20 RuleNodes in the system in a (traditional knowledge engineering)

top-down fashion after consulting with User 6. In contrast, user 12 was involved in very few case creations.

Figure 50: User 12 (the author)



Note that the initial knowledge base activity by user 12 parallels that reported in the early days of the PEIRS trial. In PEIRS the first 198 rules were added off-line while interfacing problems were sorted out (Kang, 1995, p 34-35) (Edwards, 1996, p89). In FastFIX, the first 21 RuleNodes were added in this manner.

12.10 Managing the Complexity for Users

Users were able to add RuleNodes to the system in the conventional case-based RDR manner (previously described in section 11.7.6 on page 214), or in the novel top-down rule-based manner provided by FastFIX (previously described in section 11.3.3, page 186). This meant that in terms of RuleNode acquisitions, system interaction was no more difficult than for previous conventional RDR systems, since users could always just add their new knowledge

in the local context of the case on hand, without having to understand or have any knowledge of the decision tree, or any other RuleNodes in the system.

On the other hand, the system was more flexible than conventional RDR systems, since if users did have this more global knowledge, they could capitalise on it, and enter it in a top-down manner.

As noted for Figure 25 (page 180), the hyperlinked internet-based navigation structure simplified rule tree navigation for the users. The FastFIX application allowed users to easily toggle between a case-based view with hyperlinks to the related (live or registered) RuleNodes, or a RuleNode-based view with hyperlinks to the related (live, registered or dependent) cases as well as any related (ancestor, sibling, dependent or referenced) RuleNodes.

12.11 Collaboration and Conflict

The negotiation and conflict resolution facility supported by the public (live) and private (registered) views in FastFIX were key in allowing users to negotiate and resolve their classification conflicts. As reported in section 12.2 on page 223, in total there were 107 RuleNode creations resulting in 104 case drop-through events, and 141 RuleNode edits resulting in 96 case drop-through events. Each of these case drop-through events involved 1 or more cases (and hence users). Each of the 200 case drop-through events over the course of the 172 case and 107 RuleNode software trial reported on a situation in the knowledge base system where the live versus registered RuleNodes for one or more cases (and hence users) lost synchronisation due to knowledge and hence user conflict. Affected users were notified and prompted to work together to resolve the highlighted conflicts in the acquired knowledge. Figure 110 on page 434 shows a summary of cases where the difference between the live and registered RuleNodes for each case can be observed. This represents a situation in which there is conflict, and hence a corresponding opportunity for knowledge to be acquired.

An example of a conflict that occurred at the commencement of the trial was that numerous of the RuleNodes that had been initially entered into the system used the rule "locate(0F.XX2F.XX…" instead of the more general "locate(XX.XX2F.XX…" . The success of conflict resolution between experts was anecdotally commented on by HTG staff involved with the FastFIX software trial and the rapid domain coverage reported on in section 12.5 (page 227) offers support for the approach.

The trajectories of RuleNode Creations versus RuleNode Edits displayed previously in Figure 43 (page 227) show that over time within the same domain, the nature of the conflict moved from being a conflict over the rules that defined each classification i.e. fixes to any over-generalisations and / or over-specialisations within the decision tree; to refinements of the wording of the classification labels, and the wording of the conclusions offered by those classifications. Correcting minor spelling errors was the major reason for numerous of the RuleNode Edits towards the end of the software trial in the initial problem domain.

Further examination of how experts who don't agree represent their differing points of view within the proposed system may be of interest to future researchers. When it comes to labelling controversial classifications or conclusions, a "neutral point of view" approach is recommended[184]. Note for example that some of the more controversial and debateable articles in Wikipedia (Wales, 2001) notoriously suffer from oscillations between conflicting viewpoints. Since the knowledge acquired by FastFIX is provided by human users, it seems reasonable to expect that these oscillations will at times happen for rules, classifications and / or conclusions within the FastFIX framework, and the system will need to continue dealing with the ongoing and inevitable clarifications and refinements that ensue.

12.12 Inadvertent Gross Errors

FastFIX opens the possibility of users inadvertently making gross errors, for example in RuleNode creations, deletions, edits, or relocations; that may impact on a very large number of cases. One example of a RuleNode edit that (for better or worse) impacted numerous cases during the FastFIX software trial is demonstrated in Figure 43 (page 227) at around the 450th KA event. FastFIX relies on peer pressure in a closed system, and change histories to track and manage such gross changes. In a commercial system, an "undo" feature would obviously be beneficial, as well as a "look-ahead" feature that predicted and warned the user of the impact of a proposed change prior to the user's final confirmation and commitment of the change. In the FastFIX trial, it was sufficient to re-edit any incorrectly changed RuleNodes. The idea of untracking disused cases to ensure that the impact of evolving knowledge is restricted to cases of only present and future concern is presented later in Appendix O.4 (page 454).

[184] The discussion on journalistic objectivitity at: http://en.wikipedia.org/wiki/Neutral_point_of_view is relevant.

Note that inadvertent gross errors that impact many cases also occur in conventional RDR systems. As with humans and systems that model their knowledge, gross repetitive errors can and do frequently occur. The best KA system would keep correct knowledge static while enabling incorrect knowledge to be identified and changed as quickly as possible. The idea in FastFIX is to expose the errors publicly across many users so that they are quickly picked up on and resolved. In that way the solutions are shared and remembered so that errors seldom (if ever) recur. The idea of approving specific sections of the knowledge to support confidence in the knowledge for users and to stop incorrect or immature data being used by users is presented later in Appendix O.5 (page 456).

Related to this discussion, the idea of enhancing robustness through inference is discussed in section 13.5 (page 256). As well, verification in knowledge bases has previously been proposed by Preece and Shinghal (1994) and is discussed further in relation to the Prudence work of Edwards (1996) in section 13.8 (page 263).

12.13 Knowledge Engineering Effort

To set-up this software trial, only 5 attributes were required to initially model the domain. However, as mentioned previously (section 12.3, page 225) 2-3 person weeks were spent in developing the much needed locate() function. This represented a significant KE effort, even in the simplified world of RDR.

User 12 (the author) created the first 20 RuleNodes in a top-down rule-based manner after consulting with User 6. As well, the number of case edits was 139 and the number of RuleNode edits was 141 demonstrating both the desire and capacity of users to contribute to knowledge evolution in a top-down rule-based manner.

In total 172 cases, a further 87 RuleNodes (in addition to the initial 20), and 9 attributes were captured during routine system use.

12.14 Chapter Summary

In summary, the results of the FastFIX software trial indicate that:

- Significant KE effort is still required to model the domain, create suitable differentiating functions, and pre-populate the KBS with background knowledge, even in the simplified world of RDR.

- Users were able to use the system with minimal training and supervision (less than 60 minutes on average per contributor).
- The system was able to support multiple users in collaboratively building the knowledge base, and was effective at communicating to colleagues when changes to the knowledge base occurred that might affect areas of the knowledge base that they had been working on.
- Case drop-throughs occurred frequently (in this example, about as frequently as RuleNodes were created and edited), so it appeared to be an important enhancement to separately track the live versus registered RuleNodes for users.
- Given that case drop-throughs occurred so frequently, if cornerstone case are important to users, it will be important to allow more representative cornerstone cases to be substituted for less representative ones when case drop-through occurs.
- Users were willing and able to work within both a case-based bottom-up and rule-based top-down mindset to edit RuleNodes, and to add knowledge to the decision tree.
- Users took advantage of the ability to label the classification represented by RuleNodes as distinct from the conclusions at that RuleNode.
- Users found cause to refer to conclusions at other RuleNodes via the refer() function, even though this feature was only made available at a late stage in the software trial.
- The ability to combine text and hyperlinks in the conclusions at RuleNodes meant that multiple conclusions could be referred to at a single RuleNode.
- Users found cause both to disable some RuleNodes, and to negate the effect of parent RuleNodes under certain rule conditions.
- The system was effective at rapidly acquiring sufficient knowledge to improve the effectiveness and efficiency of trouble-shooting drive-queue dial-home problems.
- The system provided a way for users to effectively “work-up” their problem cases. Users were able to add new attributes to the system.

The next chapter details additional recommended design enhancements for future system embodiments.

CHAPTER 13: DESIGN ENHANCEMENTS

13.1 Chapter Outline

This chapter details some suggested design enhancements for the 7Cs system, including:

- The idea of allowing case-RuleNode associations to be updated in a RuleNode view that reciprocates the KA facilities provided in the case view;
- The idea of either merging or rolling up RuleNodes into a SuperNode;
- A shared child RuleNode framework;
- An enriched mechanism for referring to classifications at other RuleNodes;
- A mechanism for acquiring ontological data, and representating the data at RuleNodes in a frame-based ontological style; and
- The idea of using Functional Exception Prudence to help the system recognise cases that are outside of its experience in a particular context, and warn the user of possible errors in the interpretation.

In addition a number of implementation enhancements are suggested in Appendix O (commencing on page 451).

Together with the implementation enhancements, the design enhancements outlined in this chapter will be important for ensuring the future applicability, robustness, effectiveness, scalability, and efficiency of the 7Cs design across a wide variety of problem domains.

The ideas presented in this chapter are for particularly interested readers. The usefulness and usability of many of the future design enhancement ideas presented here and in Appendix O are yet to be determined. Less interested readers should feel free to skip ahead to the Thesis Conclusions presented in Chapter 14 (page 267).

13.2 Reciprocity between the Case and RuleNode Views

Boose et. al (1992, p2-6) discuss effective mediating representations for knowledge, and they describe how different user views provide different data model perspectives for users. They discuss the importance of the different representations making important things explicit and hiding any unnecessary detail and they discuss the examples of case, decision tree, rule and

frame-based views. Lee and Compton (1995) go one step further and suggest the interlinking of heuristics (surface knowledge) with causal models to provide users with a deeper system understanding that may improve their diagnostic capacity. Similarly, Richards and Compton (1997) and Kim (2003) discuss formal concept analysis (FCA) (Wille 1992) as a mediating representation for RDR systems. The use of FCA as a mediating representation for RDR systems was noted previously in sections 4.3.2.3 (page 58) and 5.2 (page 66).

In future system embodiments, cases could simply be accepted or rejected from a RuleNode View in a reciprocal way to the manner in which RuleNodes are accepted and rejected from the Case view. For example for each RuleNode, the following RuleNode View could be provided:

Figure 51: Sample View for RuleNode N

Case	Workflow Status	Description	Live	Registered	Accept	Reject	Do Nothing
2	Assigned	cat	yes	-	●	○	○
3	Assigned	dog	yes	-	●	○	○
48	Assigned	fish	yes	-	●	○	○
57	Closed	bird	yes	yes	○	○	●
63	Closed	elephant	yes	yes	○	○	●
74	Opened	giraffe	-	yes	○	●	○
93	Closed	zebra	yes	yes	○	○	●
107	Opened	tortoise	yes	-	●	○	○
164	Verified	snake	yes	yes	○	○	●
166	Opened	eagle	yes	-	●	○	○
168	Opened	lizard	yes	-	●	○	○

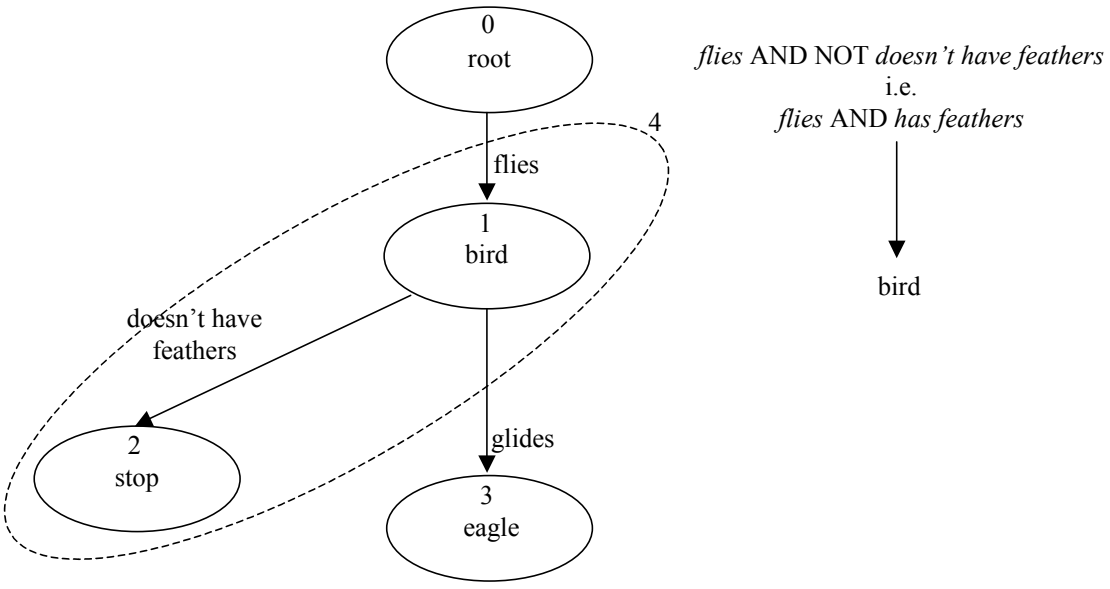
Defaults for when to accept / reject or do nothing would apply reciprocally for cases as shown in the figure above, depending on whether they are live or registered for this RuleNode, and possibly depending on their workflow status as described in section 11.5 on page 199.

As for RuleNodes in the Case View, Cases in the RuleNode View could be accepted or rejected on a Case-by-Case basis (see section 11.5.1 on page 202). The result of rejecting a Case-RuleNode association in the RuleNode view would be as for the Case view as described in section 11.7.6 on page 214.

13.3 Rolling Up RuleNodes

In my view, it would be worthwhile to allow sets of RuleNodes to be *rolled-up* into each other. For example, classifications and conclusions at dependent RuleNodes may be merged with or "rolled-up" with the classifications and conclusions at the parent RuleNodes. A SuperNode could be created that forms a grouping around two or more inner child RuleNodes. The LCL, RCL, and DCL would then be tracked for the SuperNode instead of, or as well as for its inner RuleNodes, depending on the user's preference. Figure 52 provides an example.

Figure 52: Rolling-Up RuleNodes



In Figure 52 RuleNode 1 was clarified by RuleNode 2 with an exception condition: *doesn't have feathers*. This was done to stop aeroplanes being classified as birds. When an eagle case was presented to the system, it was correctly classified as a bird, but the user wanted to provide a more specialised classification for it, and decided that compared to all the bird cases already seen by the system at RuleNode 1, it was really because it glides that it was an eagle.

A problem arose because now aeroplanes that had previously been stopped by RuleNode 2 were falling into RuleNode 3. So the user rolled-up RuleNode 2 with RuleNode 1 to create SuperNode 4 that for the cases thus far seen by the system, more accurately defines a bird as something that flies and has feathers. The user can either merge RuleNode 2 into RuleNode 1, or else maintain RuleNode 4 as a SuperNode comprised of these two inner nodes. The advantage of former approach is the use of less system resources. The advantage of the latter approach is that all the exception cases such as aircraft will continue to be tracked by RuleNode 2.

Where SuperNode 4 is used, a new case that is live for RuleNode 1 will be displayed as being live for RuleNode 4. Registration of the case will still be against RuleNode 1, but will be displayed as a registration against RuleNode 4. A new case that is live for RuleNode 2 will be stopped by that RuleNode and hence for most users, that RuleNode won't be displayed in the case view. Only if someone drills into the rule view of RuleNode 4 to view RuleNode 2 will they discover all of the cases that are currently live for it.

This feature will be helpful in managing exceptions to parent RuleNodes that have non-stopping dependent child RuleNodes that need to share the same exceptions. Previously the rule condition at RuleNode 2 did not stop cases from falling through to RuleNode 3, but via a merge of RuleNode 2 into RuleNode 1, or via SuperNode 4 it now does.

The system could allow the conditions proposed for stopping child RuleNodes to be automatically merged i.e. rolled-up into their parent RuleNodes as soon as they are created. As mentioned earlier, this need not have a negative impact on the statistics i.e. credentials maintained for the parent RuleNode. As mentioned in section 11.2.2 on page 181, the overall credentials for any given RuleNode simply needs to be comprised of a change history of the subsets of credentials recorded throughout each phase of that RuleNode's life.

13.4 Shared Child RuleNodes

Although MCRDR was very successful in reducing the replication of earlier SCRDR systems observed in multiple classification domains (a 7 fold reduction in redundancy was reported for the arterial blood gas domain), Edwards notes that even in the MCRDR implementation of PEIRS: "*Each conclusion may appear at multiple sites on the tree. Cases may therefore satisfy more than one rule associated with the same classification, and thereby generate the same conclusion by a number of paths. Clearly each conclusion only needs to be presented to*

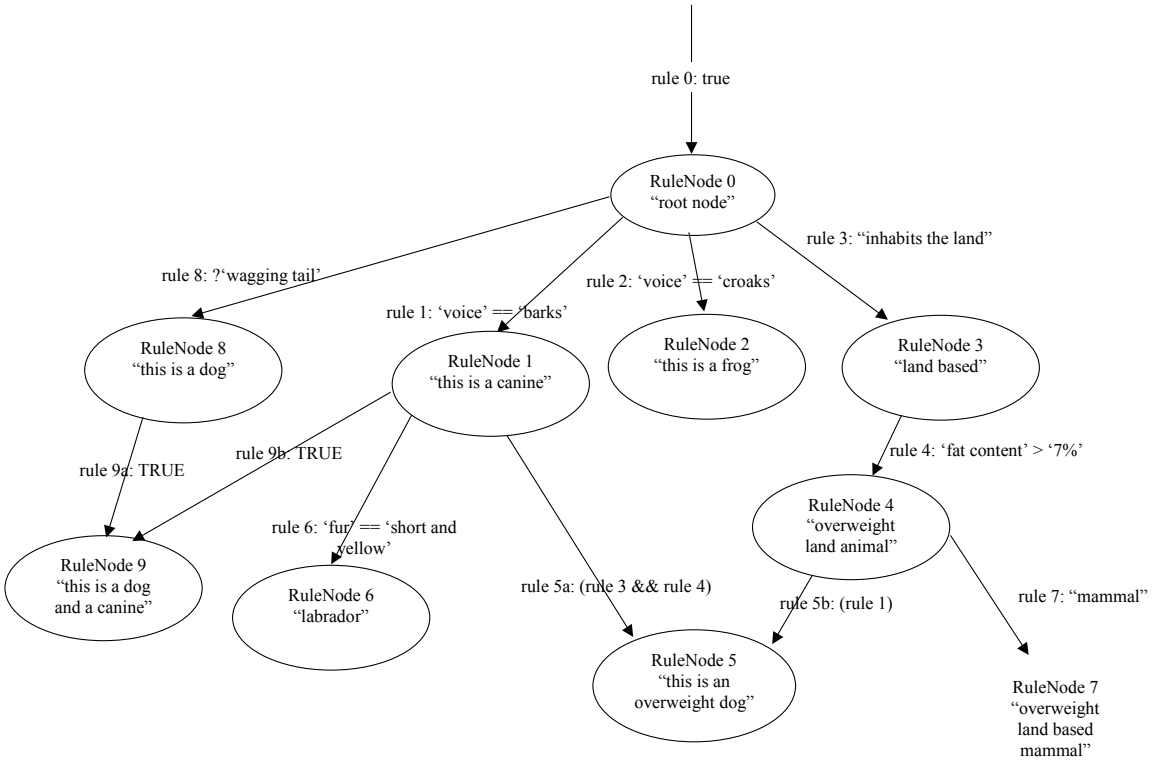
the user once… however… if rule traces are to be used for explanation, the user may wish to access each rule trace, for each classification, individually." (Edwards, 1996, pp 186, 192).

As mentioned previously in section 8.3.10 on page 155, Edwards showed that the average replication in the MCRDR worsened in proportion to the number of cases seen by the system. Given that the major problem for staff was in deciding on the classification, its wording, and the most helpful form of interpretation to provide (Kang, 1995 pp 40, 129), failure to recognise re-use opportunities for conclusions can contribute to a significant level of redundant KA in an MCRDR system. As well, over time, each replicated conclusion in an MCRDR system can result in replication of an entire sub-tree of qualifying child RuleNodes.

The appropriateness of treating identical conclusions at different RuleNodes as identical classifications is exemplified by Edwards' (1996, p202) implementation of Feature Exception Prudence (FEP), described further on in section 13.8 on page 263. He noted that there was little value in considering separately all the different RuleNodes arriving at the same conclusion, so his FEP data was provided on the basis of conclusion-based context profiles. Interestingly, Golder and Huberman note that together with polsemy and varying user requirements for specificity, synonomy is also one of the major problems in tagsomonies (2005, p2).

As shown in the following figure, a structure could be introduced where child RuleNodes can have multiple parent RuleNodes.

Figure 53: Illustrating Shared Child RuleNodes



This structure is particularly important for bringing together alternate classification paths that arrive at identical conclusions in the decision mesh. We can expect this to occur in a collaborative classification system since different users of the system with differing perspectives generate the classification pathways. For instance, one user might create a pathway saying that a dog is an animal that barks, and another user might generate a pathway saying that a dog is an animal that wags its tail. The shared child RuleNode structure is analogous to multiple inheritances in ontology, or in a frame-based or object-oriented class structure. It creates a decision mesh that is analogous to a neural network as described in section 7.2.3 (page 97). Dazeley and Kang (2004, p5, 6, and Figure 3) have previously suggested the use of a neural network as a back-end to a single-user MCRDR structure. This research proposes the loosening of the MCRDR structure to mimic a neural network that can be collaboratively developed using the live versus registered case-RuleNode paradigm introduced earlier.

In Figure 53, RuleNode 9 is a shared child RuleNode that implements an OR-ing structure for RuleNodes 1 and 8 since these two RuleNodes arrive at synonymous conclusions. The structure solves the problem of synonymous or identical conclusions being offered up in

multiple parts of the rule tree – the structure corrects the rule tree and eliminates the need for actions that modify the outputted rules to remove duplicate conclusions as described in the PKS interview in Appendix G.12 (see the comment "*Hiding of multiple identical conclusions instead of duplicate display.*" on page 411). The knowledge base could be compacted by eliminating RuleNodes 1 and 8, hanging RuleNode 9 off RuleNode 0, and changing its rule to (rule 1 OR rule 8). Automatic compaction of the rule tree may also be of value.

Note that (Compton, Cao and Kerr, 2004) have previously proposed that incremental knowledge acquisition be generalised by attending to the relations of sequence and correction. They suggest that "*systems can be recursively structured so that all KA is explicitly achieved by adding a KBS / program / rule to augment or replace output*" (Compton, Cao and Kerr, 2004, p3, paragraph 2). In this approach, knowledge bases and hence RuleNodes should never be modified, only ever added to. The authors "*hypothesise that there can be no advantage in carrying out these tasks in an implicit way by editing the knowledge base*".

In contrast what is proposed here is that shared child RuleNodes can be modified, moved or augmented using either a bottom-up case-based conventional MCRDR approach or a top-down rule-based approach, as for any other RuleNode in the 7Cs decision tree. As well, links from a child RuleNode to a parent RuleNode can be created or broken as appropriate by the users[185]. Finally, the 7Cs approach makes the 1-to-many relationship between a child RuleNode and its parent RuleNodes explicit. No extra add-on KBS is required[186].

Ontology, natural language processing and descriptive logics may be able to infer opportunities to combine RuleNodes in this fashion. The system could recognise synonymic classifications or conclusions being applied to other RuleNodes and it could proactively prompt one or more users to consider modifying an existing RuleNode instead of creating a synonymic definition. As well, the system could provide a query to find all synonymic RuleNodes and the user could unconditionally combine those paths with a shared child

[185] Note that before the system allows any parent-child relationship to be created, an acyclic implementation could check that a continuous dependency loop cannot be established where for instance a dependent (e.g. child, grandchild etc) node becomes its own ancestor node through any path.

[186] Note that the purpose of the shared child RuleNode structure is different to the purpose of intermediate conclusions in N-RDR systems. The setAttribute() feature described in section 11.3.6 (193) is of more relevance to the concept of intermediate conclusions.

RuleNode. Note that an unconditional RuleNode combine means that the rule paths from the parent RuleNodes to the shared child RuleNode are all TRUE.

Note that as for RuleNode 5 in Figure 53, the shared child RuleNode may have siblings at different levels in the rule tree according to each different parent node.

In the shared child RuleNode data structure, every RuleNode has zero or more duplets of <parent RuleNode and a rule leading from that parent RuleNode to this RuleNode>, and zero or more child RuleNodes. The following table uses this new data structure to describe the rule tree shown in Figure 53.

Table 22: Child RuleNode data structure

RuleNode	<Parent RuleNode, Rule> list	Child RuleNode list
0	<n/a, 0>	1,2,3,8
1	<0, 1>	5,6,9
2	<0, 2>	
3	<0, 3>	4
4	<3, 4>	5,7
5	<1, 5a>, <4, 5b>	
6	<1, 6>	
7	<4, 7>	
8	<0,8>	9
9	<8,9a>,<1,9b>	

With this data structure, processing commences at the root RuleNode. For each RuleNode that evaluates to `TRUE` for the case on hand, the case is evaluated against each of that RuleNode's child RuleNodes as identified in the child RuleNode list shown in the above table. The rule for each ChildNode depends on which parental rule path the evaluation is coming through. The <Parent RuleNode, Rule> list for each child RuleNode records which rule should be evaluated for each parental path. If a rule evaluates to `FALSE` then no further processing is required through that path. The set of last `TRUE` RuleNodes on each path through the rule tree constitutes the live RuleNodes for the case.

For example, referring to the table above, if the case evaluates to `TRUE` for RuleNode 1 then it is evaluated against RuleNode 1's child RuleNodes 5, 6, and 9. The system looks-up the

rule for parent RuleNode 1 at child RuleNode 5 and sees that rule 5a is the appropriate rule to evaluate for this particular path through the rule tree. The system evaluates rule 5a and if it is `TRUE` for the case on hand then processing will continue at RuleNode 5 for this case. Otherwise RuleNode 1 will be live for the case. RuleNode 5 may be evaluated as live for the case through multiple paths in the system, but it is gets recorded once that RuleNode 5 is live for the case. Similarly if a user registers RuleNode 5 for the case it need only be registered once.

The normal user accept/reject mechanism for RuleNodes applies for systems using a shared child RuleNode data structure, except that there could be an additional option in the case view to combine some live RuleNodes. There would also be an additional option in the rule tree view to combine some RuleNodes.

Note that the DCL can be constructed in the normal fashion at any RuleNode in the shared child structure. Note that if the DCL is being constructed from the LCLs of dependent RuleNodes, then in a shared child RuleNode structure a check should be made to ensure that each dependent RuleNode is only processed once.

Note that software source control offers a useful analogy for a rule tree or rule mesh. For example, Rational Rose's Clearcase source control product and the GNU freeware Code Versioning System (CVS) allow users to merge code fragments that have been evolved in parallel with other team members into a common source code via a multitude of different evolutionary paths.

Note also that the shared child RuleNode structure will require a new approach to decision mesh visualisation than that offered in the FastFIX prototype for the simpler decision tree data structure.

13.4.1 Ordered Execution of RuleNodes

The importance of controlling the order of rule evaluation was previously mentioned in section 4.3.2.3 (page 58) and is also noted in (Compton, Cao and Kerr, 2004). The order of execution and presentation for decision meshes that use this new data structure (with or without the presence of any shared child RuleNodes) can be defined by ordering the "Child RuleNode list" at the parent RuleNode. For example, the user could redefine the child RuleNode list for RuleNode 0 defined in Table 22 as {3,2,1,8} instead of {1,2,3,8}. Processing and presentation of the child RuleNodes for RuleNode 0 would therefore be done

in the order of RuleNodes 3 then 2 then 1 then 8. This mechanism could be used to manipulate the order of RuleNodes being presented to users, and hence their classifications and conclusions.

13.4.2 Merging Conclusions

Another useful application of this data structure may be in merging the presentation of classifications or conclusions. For example, a set of RuleNodes with rule paths (A, B, C) can be combined with a shared child RuleNode that tests for parent A the conditions B&C, for parent B the conditions A&C, and for parent C the conditions A&B i.e. the conjunction of the rule paths for each of the other RuleNodes. The conclusions from each of these paths can be referred to at the shared child RuleNode using the refer() function, in the desired display order, for example refer(C), refer(B), and refer(A). RuleNode 5 in Figure 53 on page 247 provides an example of this type of application, although the refer() function has not been used in that example.

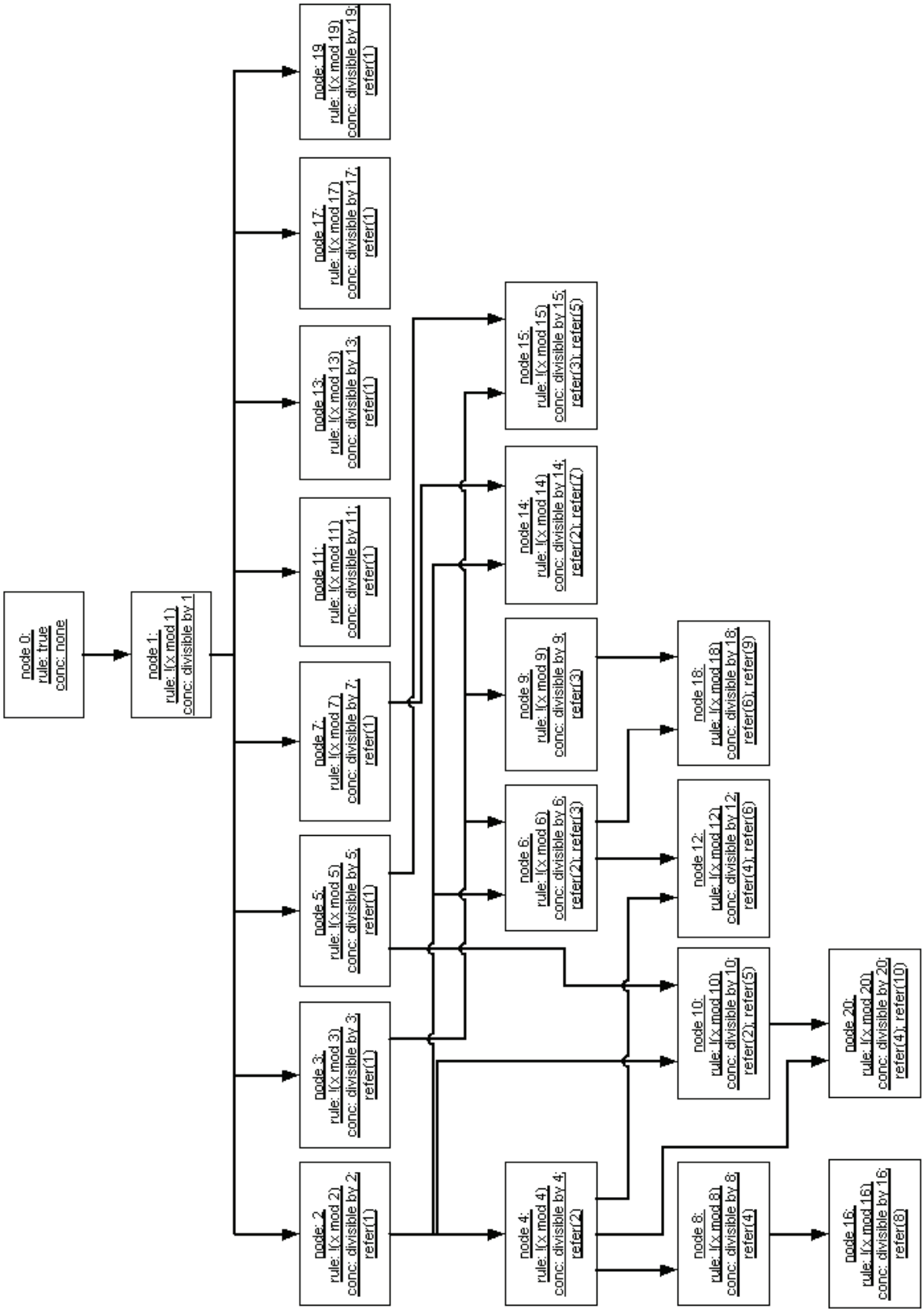
Edwards highlights the complexity of representing some combinations of disorders in chemical pathology with his example of *respiratory acidosis* and *metabolic alkalosis* (Edwards, 1996, p192, 194). He notes: "*while each may occur alone, it is debateable whether their occurrence together should be considered as two separate disorders or as a single entity*". The shared child RuleNode structure would solve this problem by allowing the disorders to be merged into a shared child RuleNode, while retaining links to and referring to the relevant parental disorders.

13.4.3 Managing Subsumption - Inheriting Classifications

A different example is provided in Figure 54. In this example, RuleNodes have been acquired for the first 20 integers to check whether or not they are prime, and to indicate their divisors. When the most recent case with integer 20 was added to this structure, the last true RuleNodes through any path of the decision mesh were RuleNodes 4 and 10. Since RuleNode 4 and 10 both refer to the conclusions at RuleNode 2, and RuleNode 10 refers to the conclusions at RuleNode 5, and since RuleNodes 2 and 5 in turn refer to the conclusion at RuleNode 1, we are informed by the decision mesh through RuleNodes 4 and 10 that integer 20 is at least divisible by 10, 5, 4, 2, and 1.

Note that user preferences could be used to determine the manner in which the inherited conclusions are displayed, and whether or not duplicate inheritances like that of RuleNode 2 (through both RuleNode 4 and RuleNode 10) are displayed.

Figure 54: Primes and Divisibility



The shared child RuleNode structure allows us to efficiently offer the advice that RuleNode 20 is also divisible by 20 by creating RuleNode 20 as a child of RuleNodes 4 and 10, and by referring to the conclusions provided by RuleNodes 4 and 10 at RuleNode 20. This allows a much more compact and useful representation than that allowed by the traditional MCRDR data structure, since the relationships between interdependent RuleNodes is utilised and maintained. The maintenance effort for this structure is arguably much reduced than for traditional MCRDR systems. The resultant class hierarchy means that duplicate conclusions and redundancy are avoided. As well, the structure is much more efficient since dependent RuleNodes are only tested for a case if the more general parent RuleNodes are already satisfied by the case. At present, there is no evidence to suggest that this exercise requires any more KE effort than that already demanded by RDR (described previously in Chapter 8, commencing on page 133).

13.4.4 Child RuleNodes with Truthful Parents

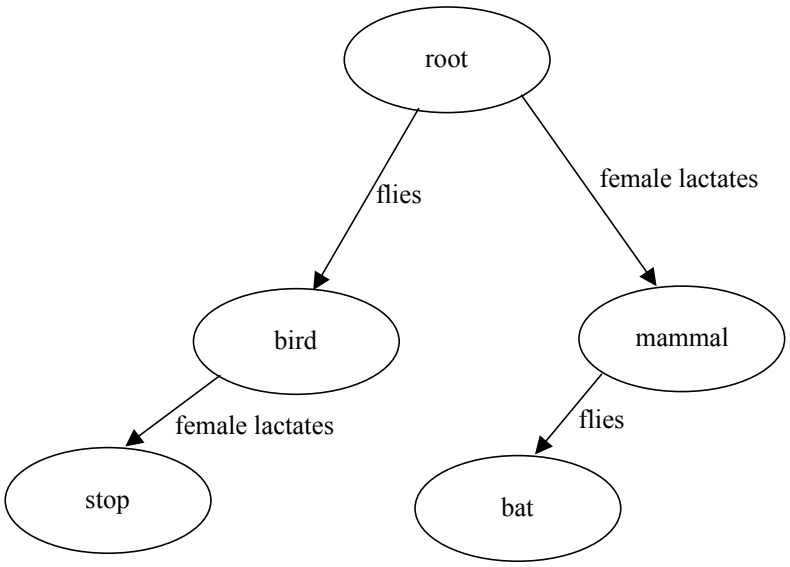
Active classifications at child RuleNodes could be restricted so that they are always specialisations of their parent RuleNode classification. Exceptions would still be permitted, but they would always be stopping RuleNodes, or incorporated as conjunctions in the condition statements of the existing RuleNodes. The actual classification and conclusions of the exception RuleNode would be located elsewhere in the decision mesh. The parent RuleNodes and their conclusions would always be true for their child RuleNodes so the refer() function wouldn't need to be used at the child RuleNodes to refer to their ancestor RuleNodes since those references would be implicit.

An example is provided in the following figure. In this example, the user asserted that *anything that flies is a bird*. Then they received a *bat* case, for which they observed that unlike birds the *female* lactates, so they created an exception condition for birds that *if it flies, and the female lactates, then it's not a bird*. Child classifications would only be allowed where the parent classification also applies. New independent classifications would need to be appropriately located elsewhere in the decision mesh. So the user would need to create the mammal and bat classifications elsewhere in the decision mesh under their relevant parent classification(s).

Note that this example was constructed using a classification scenario presented by Yao et al. (2005). Yao et al. promote the use of Ripple Down Rules for security information analysis.

In their analysis, rules succinctly summarise normal situations, and exceptions characterise abnormal situations. They argue that rule + exception strategies strike a balance between simplicity and accuracy and lend themselves to incremental knowledge acquisition.

Figure 55: Birds with exceptions



13.4.5 Matching Classifications to their RuleNode Locations

A problem arises in the previous example in the case of emus and penguins, since these are both birds, but they don't fly. In order to access the potentially rich pool of information provided by the bird classification in this example, the user of the 7Cs system would be encouraged to edit the rule condition and conclusion for bird to say for example, *if it (flies OR has feathers) AND the female doesn't lactate, then it's a bird*.

However, conventional MCRDR did not allow RuleNode editing, and it didn't allow the ORing of conditions within RuleNodes (Kang, 1995, pp 58, 36). Hence conventional MCRDR would have required that new RuleNodes be incrementally created for both penguins and emus. The original bird RuleNode would continue to create false negatives for example with Ostrich, and the relationship between ostriches, penguins, emus and other birds in the system would not have been recorded.

The functionality afforded by conventional MCRDR is unnecessarily restrictive, especially for experts who know their domain of interest well.

In conventional MCRDR implementations a choice is made whether to locate rules at the top or bottom of the rule tree. As mentioned earlier (section 8.3.8 on page 151), locating rules at the top of the rule tree tends to create a problem of false positives because of over-generalisation, and locating rules at the bottom of the rule tree tends to create a problem of false negatives because of over-specialisation. The point is that to avoid errors in either form, the classification and conclusions given at each RuleNode must correspond exactly with the rule conditions for that classification.

At the time that conventional MCRDR was developed, the following viewpoint prevailed, and possibly still prevails today (Kang, 1995, pp 62 - 63):

> *"(In) attempting to decide whether a classification is best seen as a refinement or an independent classification and the old classification stopped, we note that in some ways it does not matter – both are workable solutions for any classification."*

Unlike conventional expert system methods, the proposed 7Cs system allows context specific knowledge to be acquired in an incremental bottom-up fashion by attending to real world cases. Unlike conventional RDR methods, allowing the RuleNodes to be editable, moveable, deletable, roll-up-able, and replaceable in a top-down KA manner, lets users optionally attend to inconsistencies between classifications and their rule conditions, and to improve the performance of the expert system by tightening the knowledge representation. Further to this, allowing the conditions at RuleNodes to be ANDed, or ORed allows users to reduce both false positive and false negative errors. Finally, the shared child RuleNode mechanism assists users in creating a knowledge base with less redundant and duplicate knowledge.

13.5 Enhancing Robustness through Inference

The idea of using knowledge histories and inference to build more robust expert systems has been previously highlighted by Kang (1995, pp 121, 122, 138, 140) and Horn (1993, sections 10 and 12). A large part of the description logics effort described in section 5.6 (page 76) is also targeted at inferring knowledge in KBSs.

An example of where the system can infer opportunities to combine rule paths in conventional MCRDR systems is given by Kang (1995, pp 70, 71[187]). In systems with a shared child RuleNode structure that demonstrate this concatenation pattern, combining RuleNodes can be to be done by inserting a RuleNode directly beneath the RuleNodes that share the classification. Stopping child RuleNodes that were previously dependent on either of the two RuleNodes that shared the classification should be rolled-up into those RuleNodes. Other child RuleNodes that were previously dependent on either of the two different RuleNodes that shared the classification need to be moved to become dependent RuleNodes of the new shared child RuleNode.

A second inference opportunity is highlighted by Kang where two different paths result in a mirrored ordering of the same classifications e.g. classification A followed by sub-classification B and in an alternate path, classification B followed by sub-classification A (Kang, 1995, p71)[188]. A control study by Kang (1995, p123) showed that this type of classification mirroring occurred in more than 10% of classified cases in the GARVAN-ES1 system. These paths could be combined and both sets of classifications and conclusions could be offered at the shared child RuleNode.

Another inference opportunity is where equivalent sets of rule conjunctions arrive at different classifications and conclusions. Again, combining RuleNodes could be done by rolling-up any stopping RuleNodes into their parent RuleNodes, then inserting a RuleNode directly beneath the RuleNodes that share the classification and moving any previously dependent RuleNodes underneath the new shared child RuleNode.

The idea of separately tracking the live versus registered case-RuleNode associations and applying induction to refine the acquired knowledge is supported by previous evaluations by Kang. As a result of his controlled study of the GARVAN-ES1 system, Kang concluded that a history of changes in knowledge may be used as an important source of expertise, or rather common sense prudence for KBS (Kang, 1995, pp 120-126).

[187] See pattern 1: basic, and pattern 3: concatenate.

[188] See pattern 2: mirror.

13.6 Using Classification Labels

13.6.1 Moving from the General to the Specific

Problem determination is both a configuration and a classification problem: configuration is required to "work-up" the case, and classification is required to "determine the class of problem on hand". At the highest level, when the least is known about the problem case on hand, general solutions will apply such as "search the Internet", or "read this user guide". As more is known about a problem, more specific solutions become relevant such as "look at this particular website" or "read this paragraph on this page of the user guide". The traditional structure of MCRDR does not handle the transition from general to specific all that well since users must either accept both the general and specific conclusion, and risk being overwhelmed by conclusions; or reject a true but more general conclusion and replace it with a more specific conclusion for the given problem class. In the call centre context, the amount of information that users need to wade through needs to be minimised, hence only the most pertinent and relevant solutions should be presented.

A control study by Kang showed that both general and more specific classifications were ambiguously derived for more than 40% of classified cases in the GARVAN-ES1 system (Kang, 1995, p123).

The following table lists four ways in which new RuleNodes correct a knowledge base and in particular introduces a new way, Scenario D, in which RuleNodes may be added by a user to correct a knowledge base.

Table 23: Four Ways in which new RuleNodes correct a KBS

Scenario Identity	Description	Conclusion Status	To correct the KB
A: Reject	Reject a classification because the classification does not apply	Incorrect	Add a stopping RuleNode at the end of the path to prevent the classification
B: Replace by Rejecting and Adding	Reject an existing classification and create a brand new independent classification instead	Incorrect and Missing	Add a stopping RuleNode at the end of the path to prevent the classification AND add a RuleNode somewhere else in the decision mesh to give the new classification
C: Add	Create a brand new classification	Missing	Add a RuleNode somewhere else in the decision mesh to give the new classification
D: Refine	Refine a more general classification in order to be more specific	Greater specificity desired	Add a refinement RuleNode at the end of path to provide a more specific classification.

Adapted from Kang (1995, p63).

Scenario D in Table 23 presents a new scenario not previously handled by conventional MCRDR systems. In this Scenario, the information presented is not incorrect, but rather it is imprecise, and it needs refining rather than replacing. This scenario commonly occurs in taxonomies or ontology where one moves from more general classifications to more specific classifications, for example, when moving from mammal to a dog or a cat.

In this scenario, the user can indicate that they wish to refine the information provided rather than replace or reject the information. In that case, when the user creates the new child RuleNode, the system may pre-populate the conclusion with a conclusion that refers to the parent RuleNode. This was discussed previously in 13.4.2 on page 251. The following table provides some conclusion examples for the referring RuleNode:

Table 24: Conclusion examples for the referring RuleNode

Pre-populated conclusion at the referring RuleNode	The effect that you get whenever the referring RuleNode is included in a case view
refer('mammal')	all the conclusions for the referred classification "mammal" are pre-processed and displayed in the referring classification
link('mammal') or 'mammal'	a hyperlink to the mammal classification is provided for the word "mammal" – the user can click on this hyperlink to bring up the mammal classification details or a pop-up is activated on mouse-over across the word "mammal" to provide the mammal classification details or a pop-up is created and activated through a right click menu selection on the word "mammal" to provide the mammal classification details

Similarly, if the referred classification is unnamed, a numeric reference to the referred classification can be used such as refer(43) or link(43) where these functions pre-process and extract the conclusions from the given classification (in this case RuleNode 43) as required, and display the conclusions according to the above table.

Note that this referral mechanism can also be used for arbitrarily referred to classifications across the rule tree. In other words, the conclusions for a RuleNode can refer to any other RuleNode in the system with the refer() and/or link() function.

The behavior of the refer() and link() functions for conclusions depends upon the particular implementation. Behaviors may also be defined by system and user specific configurations.

13.6.2 Copying the Details from Arbitrary RuleNodes

It can be seen in Figure 5 on page 52 that RuleNodes 4 and 5 are variations on the same theme – both conclusions recommend an upgrade with a software patch. Therefore, if RuleNode 4 was allowed the additional conclusion for example, "refer to the upgrade procedure document available at: http://myintranet/upgrade.pdf" where the latter text string is an Internet URI that the viewer can click on to be directed to the relevant document in their web browser, then it would be of benefit to allow a user creating RuleNode 5 to indicate that it is a variation on the

theme of RuleNode 4 so that when RuleNode 5 is created, the user can simply edit and modify the text provided at RuleNode 4, and apply it to RuleNode 5.

When a user is constructing a new RuleNode, a request can be made to copy the details from another RuleNode. In that case, the other RuleNode's rule and / or conclusions and conclusion types are copied to the view where the new RuleNode is being constructed, and the user can edit them to form the new RuleNode.

13.7 MCRDR and Ontology

Ontologies were previously discussed in section 3.4 (page 34) and have previously been discussed in the context of RDR for example by (Suryanto and Compton, 2001) and (Beydoun et. al, 2005).

Ontologies are typically built top-down by creating classifications from top-of-the-head expert knowledge. Farquhar et al. (1995a) have previously highlighted the opportunity to use context logic to integrate ontology data from different sources. Following this line of reasoning, an agent-driven 7Cs approach could be used to discover domain specific ontologies from the bottom-up, on the basis of cases seen, as well as using top-down background knowledge. As with RDR approaches in general, the 7Cs approach allows a mental map to be formed from the Assertional case or instance-based A-box to the abstracted terminological T-box in a manner that concludes which meronyms are contained by classes, and the relationships between classes in the system (Baader and Sattler, 2000) (Baader and Nutt, 2003).

At some point in the future the best features of manual KA and inductive machine-learning approaches could be possibly be combined to extract ontologies, for example from natural language text. Table 25 has been reproduced from Maedche and Staab (2001, Table 1) and demonstrates some current approaches to ontology learning.

Table 25: Classification of Ontology Learning Approaches

Domain	Method	Features used	Prime purpose	Papers
Free Text	Clustering	Syntax	Extract	Buitelaar [3], Assadi [1] and Faure & Nedellec [6]
	Inductive Logic Programming	Syntax, Logic representation	Extract	Esposito et al. [5]
	Association rules	Syntax, Tokens	Extract	Maedche & Staab [17]
	Frequency-based	Syntax	Prune	Kietz et al. [15]
	Pattern-Matching		Extract	Morin [22]
	Classification	Syntax, Semantics	Refine	Schnattinger & Hahn [11]
Dictionary	Information extraction	Syntax	Extract	Hearst [12], Wilks [35] and Kietz et al. [15]
	Page rank	Tokens		Jannink & Wiederhold [13]
Knowledge base	Concept Induction, A-Box mining	Relations	Extract	Kietz & Morik [16] and Schlobach [28]
Semi-structured schemata	Naive Bayes	Relations	Reverse engineering	Doan et al. [4]
Relational schemata	Data Correlation	Relations	Reverse engineering	Johannesson [14] and Tari et al. [32]

Reproduced from Maedche and Staab (2001, Table 1)
with the kind permission of Steffen Staab.

In the MCRDR implementation of PEIRS, Edwards noted that the generation of invalid combinations of RuleNodes and hence interpretations was a significant issue, and that there was no mechanism for preventing invalid RuleNode combinations from occurring (Edwards, 1996, 1996). In future embodiments of the system, the user could identify class relationships like those used ontology, for example in the RDF/OWL specifications. As suggested by Hakimpour et al. (2001, section 4) descriptive logics (DLs) could be used to check the consistency of definitions and reduce semantic conflicts. More specifically, DLs could be used to suggest opportunities to combine RuleNodes where synonyms are used; detect opportunities to merge parts of the Rule Mesh when homonyms or hypernyms are implied; and infer conflict when attempts are made to combine RuleNodes with antonym classifications or conclusions[189]. In Appendix R, some of the class relationships that could be managed are outlined, and an example is provided that demonstrates how this information could be represented.

[189] Note that at this stage I have excluded the paronym, metonymy, and exocentric semantic forms, because they don't appear to be as relevant as the forms outlined in this section. I have also left out coordinate terms (sibling classifications that share the same generalised parent).

13.8 Prudence

Preece and Shinghal have previously characterised anomalies as redundant knowledge, contradictory knowledge (ambivalence), circularity, and missing knowledge (deficiency) (Preece and Shinghal, 1994, p3 and Figure 1 on p8). They found that: "*despite the best efforts of their developers, real-world KBS often contain anomalies, sometimes in surprisingly large numbers. Furthermore, where anomalies are present, they frequently correspond to errors in the KB*", (Preece and Shinghal, 1994, p15 and Table 3). In conclusion they.found that "*integrity checks inexpensively reveal anomalies in KBs, which have a high likelihood of indicating errors*" (Preece and Shinghal, 1994, p17).

Further to this, Edwards conjectured that expert systems should recognise the limits of their knowledge and experience, and seek assistance from others when faced with uncertainty (Edwards, 1996, p199). Edwards (1996) advocated that error in KBSs should be predicted and actively managed (p197) and as a result of his experience with PEIRS, he introduced the concept of prudence in RDR systems (pp i, vi, 198, 211). To evaluate the utility of *Prudence*, he built a new expert system for the PEIRS data, based on the aforementioned TC-RDR shell (section 4.4.6, page 62).

Since FastFIX is able to track multiple cases at each RuleNode it easily lends itself to the implementation of the Feature Exception Prudence (FEP), Conclusions Prudence (CP), and Feature Recognition Prudence (FRP) features outlined by Edwards and described below. These features could be incorporated, along with the novel Functional Exception Prudence feature proposed below.

As noted by Edwards (1996, p217) some flagged features will be supported by certain conclusions and irrelevant to others (p217). To avoid information overload in the context profiles, he suggests updating the context profile at RuleNodes to note which features may be relevant in detecting FEP and CP at that RuleNode. Therefore the user should be able to filter out the irrelevant FEP and CP flags at each RuleNode, as well as filter out the irrelevant Functional Exception Prudence flags proposed here.

As noted in section 4.4.7 (page 63) later work on Prudence was conducted by Dazeley and Kang. However, the shared child RuleNode structure proposed earlier in the 7Cs model, and the idea of approving RuleNodes in Appendix O.5 (page 456), provides a viable alternative to the rated MCRDR approach.

13.8.1 Feature Exception Prudence

Feature Exception Prudence (FEP) (Edwards, 1996, pp 199-200) refers to the ability of the expert system to recognise data in cases that are outside of its experience in a particular context, and warn the user of possible errors in the interpretation. While rule conditions represent the required features for an interpretation, Edwards recognised that other data in an expert validated case constitutes the permissible features. He proposed that context-specific profiles be built at each RuleNode noting the permissible case features for that RuleNode. In acknowledgement that conventional MCRDR systems result in identical conclusions being redundantly represented across the knowledge base, his data presented context-specific profiles for each unique conclusion, rather than each unique RuleNode (pp 202 - 205).

In FEP, every time a new case evaluates to a given conclusion, the case is compared against the context profile of permissible features for that conclusion, and the user is warned of any unusual features in the case compared to the experience of cases thus far at that conclusion. Warning could be presented in the form of an out-of-range value, graphically, or statistically, for example in terms of standard deviations from the mean (p214). As more cases are seen, and the system learns from the cases that it interprets, the context profile at each conclusion can be deliberately widened by the expert updating the system. Merging of identical classifications in the proposed 7Cs system using the shared child RuleNode data structure (section 13.4, page 245) would allow FEP to be implemented on the basis of RuleNodes, rather than conclusions. The FEP profile at each RuleNode would be matured by using cases that are live and/or already registered for the RuleNode.

In his study of FEP with a subset of the PEIRS data and his early LabWizard TCRDR implementation, Edwards showed that of the 53% of the 995 cases interpreted 100% were correctly interpreted. He suggests that reports generating no FEP flags might therefore be exempted from human validation (Edwards, 1996, p211). This assumes that all the relevant case data has in fact been codified and included in the case, and that the system has a suitable way of comparing the similarity and difference between the possible values of case attributes. In the examples provided, only numeric attributes with values falling into known ranges are subjected to FEP (p204).

13.8.2 Functional Exception Prudence

It may be worthwhile evaluating the functions used at RuleNodes elsewhere in the knowledge base for the cases that are live for a given RuleNode, in order to determine the similarity and differences between possible values of the case attributes, and hence the novelty of A-V combinations from the KBS's perspective. This need is similar to the need to display the results of function calls in the difference lists of attributes. This would be particularly helpful for free text fields for example where grep() and/or locate() style functions apply.

13.8.3 Conclusions Prudence

Conclusions Prudence (CP) (Edwards, 1996, p218) refers to the novel presence or absence of other live RuleNodes for the case in question. It is very easy to implement, and would provide a subset of the more comprehensive functional exception prudence test suite proposed in the previous section. It only tests for the evaluation of functions in RuleNodes that are reachable by the case under review.

13.8.4 Feature Recognition Prudence

Feature Recognition Prudence (FRP) (Edwards, 1996, pp 207-208) refers to the ability of the expert system to recognise duplicate conclusions and hence duplicate RuleNodes in the knowledge base, and evaluate the present case against the child RuleNodes of the duplicated RuleNode to determine whether it would fetch an alternate conclusion.

Edwards notes that of the 1610 PEIRS cases evaluated by the TCRDR-based LabWizard tool that he developed to evaluate his proposed Prudence features, FRP found possible interpretations for 8 cases and 7 of the 8 alternate interpretations were found to be correct (p207).

Merging of RuleNodes with identical classifications and/or conclusions in the proposed 7Cs system using the shared child RuleNode data structure would achieve the same result as FRP, without any additional effort.

13.9 Chapter Summary

In summary, this chapter has proposed a number of design enhancements including:

- A reciprocal view of case acceptance and rejection within the RuleNode view;
- A mechanism for merging RuleNodes or rolling them up with the use of a SuperNode;

- A shared child RuleNode data structure that allows multiple paths through a decision mesh to arrive at the same RuleNode;
- A richer use of classification labels and ontological concepts throughout the decision mesh data structure; and
- The introduction of Functional Exception Prudence and other prudence features.

It is believed that these features will be important in ensuring the future applicability, robustness, effectiveness, scalability and efficiency of the FastFIX knowledge acquisition engine across a wide variety of problem domains.

The next chapter presents the summary and conclusions for the thesis as a whole.

CHAPTER 14: SUMMARY AND CONCLUSIONS

14.1 Chapter Outline

The first part of this chapter provides a summary of the body of this thesis. Following this, the main outcomes of the thesis are recalled, and relevant caveats are presented. The thesis concludes with some suggestions for future work.

14.2 Thesis Summary

Recalling from section 1.3 (page 3) that the aim of the research was to develop a knowledge acquisition approach to facilitate the capture and recall of problem solving knowledge in a dynamic multi-user knowledge environment using the HTG support centre as a case study, the questions addressed by this research were presented in section 1.4 (page 4) and pertained to 1. the nature of the trouble-shooting task; 2. problems with existing approaches to information recall and knowledge management; 3. existing software approaches to collaboration; 4. the changes required to adapt MCRDR to a dynamic and collaborative problem domain; 5. the stochastic nature of case-driven KA and a comparison with rule-driven KA; and 6. the motivations behind a hybrid case-based and rule-based KA approach. This aim and these research questions were addressed in the body of the thesis, as summarised in the remainder of this subsection.

In Chapter 2 (page 13) and Appendix A (page 305), the support centre interviews and survey at HTG revealed that the trouble-shooting process could involve complex problems, requiring a wide span of experience and knowledge from multiple different and global sources. Tacit knowledge seldom shared within the trouble-shooting group included: problem determination, where-to-search and what-to-search-for knowledge. It was found that troubleshooting involves a case-configuration-classification-conclusion cycle (section 2.5.3 on page 26). As well, it was found that significant organisational inertia was locked up in the existing case-tracking and solution-reporting systems at HTG, particularly in regard to culture, training, metrics and management reporting (section 2.4.1, page 17). As a result of the interviews and survey at HTG, a large number of system and software requirements were identified (section 2.6 on page 27, and Appendix C on page 390). Although it was beyond the scope of this thesis to satisfy all of these requirements and to thoroughly test that the requirements had been met, the requirements analysis was used to guide the design of the proposed 7Cs system.

One of the benefits of the proposed 7Cs software architecture is that through the use of URI-style hyperlinks, it can act as a very lightweight information broker that sits in-between legacy case and solution tracking systems, without requiring any modification to these databases or any major modification to an organisation's existing workflow.

In Chapter 3 (page 30) we learnt why popular search algorithms don't work on data like that observed in the HTG dial-home problem domain. We found that anchor text and hence tagging is an important asset to search engines in solving the findability problem. Folksomonies show us that it is possible to create a KBS that relies on some form of article tagging by users, and the frequency or popularity of those tags, to increase the findability of knowledge for both individuals, and groups. As well, Wikipedia teaches us that it is possible to create a KBS that lets multiple users continuously contribute to the best of their knowledge, so that the best knowledge of all users is the one that persists.

Chapter 4 (page 40) provided a review of conventional RDR systems. Some past RDR implementations and variations to the RDR theme were discussed.

In Chapter 5 (page 64), the importance of seeking input from multiple experts when attempting to elicit knowledge about a domain was discussed. It was concluded that if we allow multiple experts to collaborate in the knowledge acquisition task and resolve their classification conflicts via an expanded MCRDR paradigm, then we may be able to get the benefits of a Wikipedia-style collaborative KA forum. A solution could be developed that is analogous to the Folksomony idea where multiple users tag solutions with lexical symbols meaningful to both themselves and their colleagues. In a multi-expert MCRDR paradigm, the tags or anchor text would be comprised of the case-oriented conditions or rules that examine the attributes of incoming cases and specify when a particular solution should apply to a particular class of case. Such a solution would parallel the anchor text phenomenon exploited by search engines, where multiple unrelated webmasters inadvertently tag links to common websites with lexical symbols that correlate with the symbols applied by arbitrary users when attempting to recall that website using a popular search engine. The end result would be a case-driven collaborative classification system that could be used to index solutions so that they could be easily rediscovered when exemplar cases for known problem classes arrive.

In Chapter 6 (page 79), the idea of tracking the live versus registered case-RuleNode associations in MCRDR was presented. This would allow the impact of case drop-downs in a

dynamic and multi-user MCRDR environment to be optionally tracked, and any knowledge discrepancies optionally resolved by users at their discretion and at their convenience. The delayed validation and verification mechanism proposed by this research allows both top-down rule-driven and bottom-up case-driven changes to be made to the knowledge base in a way that promotes a context-specific negotiation of changes to take place using real examples of affected cases, at a time that best suits the users. The RDR benefits of capturing relative knowledge, in context are retained, while greater flexibility is offered to users in the manner in which they choose to contribute their knowledge (top-down versus bottom-up), what types of case-RuleNode associations they wish to track (Appendix O.4, page 454) and the time at which they choose to resolve any knowledge inconsistencies. Review of social aspects of the PEIRS experience in section 4.3.1.2 on page 48 highlighted the important of achieving local concordance with domain experts. Importantly, Edwards noted that improving the users' control over and understanding of the system would facilitate its uptake into routine practice (Edwards, 1996, pp 229, 231, 244 - 246). As mentioned in section 8.4 (page 158), validation and verification can be further supported through the notion of approved RuleNodes as described later in Appendix O.5 (page 456).

In Chapter 7 (page 92) a mathematical model was provided for case-driven and rule-driven KA that showed that the probabilistically expected trajectory for case-driven KA is comprised of a monotonically increasing and rapidly slowing KA curve. Formulas were derived and discussed for single and multiple classification scenarios, for domains with classes that occur with different frequencies, and for domains where multiple parties are transferring the knowledge. The model offers important predictions for the knowledge acquisition curves seen in previous SCRDR and MCRDR machine learning trials, as well as the tag acquisition curves observed in Folksomonies and other collaborative tagging systems. Reciprocal formulas were derived for the expected number of cases needed to acquire a certain number of classes, and the expected number of classes that would be acquired after a given number of cases has been seen.

Chapter 8 (page 133) explained how RDR systems and expert systems in general might require a significant amount of Knowledge Engineering expertise, depending on the target problem domain. Based on the availability of ground rules and background knowledge; the real nature of classification errors; the cost of correcting errors and the performance requirements of real-time interactive systems, the analysis in Chapter 8 called for a hybrid

Case And Rule-Driven (C.A.R.D.) approach to knowledge acquisition. A hybrid approach can be used to prevent errors before they occur, minimise the cost of correcting errors, and enhance the overall performance of the system.

The top level design of the proposed 7Cs system was presented in Chapter 9 (page 161), comprising the Collaborative Classification and Configuration of a stream of incoming Cases via a relational structure of ConditionNodes, Classes and Conclusions (hence 7Cs). Key features included the idea of a decision mesh with shared child RuleNodes, rather than just a decision tree; multiple labelled classifications per RuleNode; multiple conclusions per RuleNode; and multiple experts collaboratively updating the knowledge.

In Chapter 10 (page 169), the dial-home problem context was discussed and a sample CaseDB case was presented (Appendix B, page 387). The problem context demonstrated a domain that did not lend itself to the plain text keyword search algorithms implemented by popular search engines; or to the easy application of data mining or machine learning techniques. Rather it was a domain with a high volume of repetitive incoming problems that lent themselves to manual indexing via user-defined indexing functions.

The 7Cs and FastFIX design core was presented in Chapter 11 (page 176) and the FastFIX Prototype shell was introduced in Appendix M (page 424).

Chapter 12 (page 223) summarised the results of the FastFIX software trial. It was found that the 7Cs system was effective at promoting conflict resolution and hence knowledge acquisition by communicating to colleagues when changes to the knowledge base occurred that might affect areas of the knowledge base that they had been working on.

Finally, Chapter 13 (page 242) proposed some design enhancements for the 7Cs system and Appendix O (page 451) presented some implementation enhancements.

14.3 Claims and Caveats

In the three subsections that follow, each of the main outcomes of the thesis is recalled, and the relevant caveats are revisited.

14.3.1 The Nature of Trouble-shooting

The outcomes of this thesis were summarised and presented in section 1.6 (page 7). The first of these outcomes was as follows:

1. A review of the trouble-shooting environment at HTG, showing that trouble-shooting know-how includes the ability to configure (i.e. work-up) a case, classify it, and locate its relevant solution.

Some caveats and explanations should surround this claim. Although the interviews and survey presented in Chapter 2 and Appendix A were as comprehensive as time would allow, this thesis represents just one small case study in the enormous, complex and much studied support centre space, not to mention the related and broader areas of Artificial Intelligence (AI), Knowledge Management (KM), Information Systems (IS), troubleshooting, collaborative configuration, and collaborative classification in general.

Intuitively the reported case-configuration-classification-conclusion cycle makes sense. But there are many problems with trying to take heuristic knowledge (in a structured, semi-structured, or natural language form) from multiple different experts and trying to codify it in a discrete computer-storable representation[190] that makes sense to multiple different users with multiple different cultural, dialectic and experiential paradigms, and multiple and sometimes conflicting needs, including different priorities and timeframes.

As noted by (Beydoun et. al, 2005, p52) and in section 7.2.6 (page 118), the performance of the KBS depends on there being frequent exemplars of common classes. Hence the economies of scale necessary for system success will only be reached if the encoded rules act on the incoming cases in a manner that causes those incoming cases to be accurately clustered in volume into a much smaller number of repetitively represented classes. Further to this, depending on the type of problem domain, the brittleness problem attributed to conventional expert systems and referred to previously (section 3.5.1 on page 36) still applies. As Dazeley and Kang note (2004): "*it is next to impossible to include all the required knowledge to completely eradicate the inherent brittleness of these systems*". As well, the necessary attribute and function lists must be comprehensible and manageable by the users. The 7Cs and indeed the MCRDR knowledge acquisition and classification strategies will only succeed if the encoding benefit exceeds the actual cost of codification.

An initial requirements analysis was presented in section 2.6 (page 27) and further requirements were listed in Appendix C (page 390). Further to these requirements, and as

[190] Recall the discussion of analogue versus digital knowledge in Appendix E (page 404).

with any expert system, the system needs to be compelling so that users keep adding to and refining the knowledge. The resultant classifications and conclusions presented need to be accurate and relevant and the suggested solutions need to solve the target problem. Any new system has to be compelling enough to overcome the significant organisational inertia that is locked up in incumbent systems and processes within an organisation (previously discussed in section 2.4.1, page 17). Beydoun et. al (2007) recall Forsberg's (Forsberg et. al, 2001) characterisation of design issues for social software navigation systems, namely that they require integration, that a variety of users need to actually use the system, and that users need to be able to trust the information provided by the system (Beydoun et. al, 2007, p3, p11). As Dazeley et. al (2004) note, when a system is queried beyond its capabilities and it returns wrong information, it can cause users to "*lose faith in the computer's ability to give an accurate and meaningful conclusion*". Validation, verification and completeness are therefore important usability issues.

14.3.2 Modelling Knowledge Acquisition

The second thesis outcome presented in section 1.6 (page 7) was as follows:

2. The derivation of a stochastic model that explains and provides predictive formulas for Case-driven Knowledge Acquisition as in Single Classification Ripple Down Rules (SCRDR) systems, Multiple Classification Ripple Down Rules (MCRDR) systems, and Collaborative Tagging Systems such as Folksomonies.

One caveat that should apply to the analysis in Chapter 7 surrounds the nature of knowledge itself. Although the term "Knowledge Acquisition" has been used extensively in this thesis, and some discussion was presented in Appendix E (page 404) and in section 7.2.3 (page 97), the knowledge being acquired in the presented mathematical model was greatly simplified. Only the acquisition of new classifications was considered. The acquisition of knowledge about inter-relationships between the classes was not discussed. For heavily inter-related problem domains, such as word tokens in human language, the ability to acquire knowledge about the inter-relationships[191] between classes will obviously increase as the acquired number of classes increases. Further examples of this were provided in section 7.2.11 (page

[191] The increased KA complexity for domains where class inter-relationships need to be acquired was noted in point 11 of section 8.2.7 on page 139.

123). In these types of domain, the case-driven rate of knowledge acquisition could increase over time rather than decline.

Another caveat that applies to the stochastic case-driven KA model relates to whether the transferred knowledge is sticky (see section 7.2.8 on page 122) i.e. that once it has been transferred and received it is also stored and the KA process will not work to deplete it. Yet another caveat relates to whether the transferred knowledge is true, false or speculative (see Appendix Q, page 473). Obviously incorrect knowledge should result in subsequent KA activities to correct the knowledge.

As well, and as discussed in section 5.2 (page 66), users can have different needs for the acquired knowledge and those needs typically change over time. As discussed in section 11.7.2 (page 211), the separation of live and registered case-RuleNode associations in the proposed 7Cs system was a key part of being able to resolve classification conflicts between multiple experts, and between what a single expert thinks today, as compared with tomorrow (Gaines, 1993). As discussed in section 7.2.11 (page 123), the upward drift in the size of the classification domain (*m*) in the case-driven KA curves observed for Folksomony data (Figure 9, page 96) exemplifies the dynamic nature of knowledge.

14.3.3 Extracting Knowledge

The third and final thesis outcome presented in section 1.6 (page 7) was as follows:

3. A knowledge representation and knowledge acquisition technique known as 7Cs that supports the **C**ollaborative **C**onfiguration and **C**lassification of a stream of incoming problem **C**ases via a set of **C**onditionNodes linked to their **C**lasses and associated **C**onclusions.

7Cs was designed to allow a group of trouble-shooters to collaboratively configure and classify their incoming problem cases, for the most part by asking users to confirm, modify or incrementally augment the rule conditions that classify those incoming cases, and by linking those problem classes to relevant solutions that they and their peers can rely upon when repeat exemplar cases are received. The proposed 7Cs collaborative-indexing system enables trouble-shooters to share the questions that they ask themselves when classifying incoming problems, and thereby assist themselves and their peers in configuring or *working-up* their incoming cases. 7Cs acts like an organisational memory, allowing a group of trouble-

shooters to share their solutions and continuously refine the quality, relevance, and consistency of solutions offered.

One caveat that should apply to the proposed 7Cs approach is that the problem domain must be suited to a classification-based KA approach. As previously discussed (section 8.2.7, page 138), apart from the experience, capability and capacity of users, the size and complexity of the knowledge engineering and hence knowledge acquisition task is domain dependent, even for MCRDR and hence 7Cs systems. As well, in the construction of the FastFIX trial, it took 2-3 weeks to develop a suitable locate() function to help provide structure to the only partially structured case data (section 12.3, page 225). In some domains, it may not be possible to derive suitable higher-level functions to differentiate the cases. The number of attributes required to model the domain might be enormous, and the aforementioned problems associated with data mining and machine learning would apply (see Appendix D.8 on page 403 and section 3.2 on page 31).

The types of knowledge domains where 7Cs might apply was previously described in section 9.5 (page 166) and includes high-volume, complex, repetitive and codifiable knowledge domains where users rely on heuristics to form classifications. If the domain criteria summarised in section 9.5 are not met, then the 7Cs solution does not apply.

Another caveat that should apply relates to the ability of users to collaborate in providing sufficient convergence in the set of attributes that model the domain, and in the classes that divide the problem space so as to achieve a sufficient level of reuse in the classifications and conclusions. The FastFIX trial was a great start, but much more testing would be required to determine whether users on the whole could really work together in modelling and classifying a problem domain, and in resolving their inevitable conflicts[192]. As discussed in section 8.4 (page 158), a comparison of voting versus negotiating or other strategies to build consensus was left open by the research.

Further to this, there was relatively little testing of the facility for users to work-up their problem cases, and share the questions they ask themselves when classifying problems. Most of the testing surrounded the classification of previously configured cases, rather than cases configured on the fly. As well, important features like the use of shared child RuleNodes and

[192] Thanks to Debbie Richards (personal communication, 2006) for highlighting this caveat.

the facility to track and untrack cases remains as future work. Obviously much more trialling, testing and refinement would be required to convert the 7Cs approach and / or FastFIX prototype into a production and / or commercial system. In particular, the scalability of the system, the practicality of tracking possibly volumes of live versus registered case-RuleNode associations, and the implications of delaying the validation and verification of modifications to the KBS all require ongoing investigation.

14.4 Future Work and Thesis Conclusion

Areas of research left for future work were identified in sections 7.3.4 (page 127) in regard to hypothesis testing with the derived case-driven KA model; and in section 8.4 (page 158) in regard to testing the domain-dependent size and complexity of the KE and hence the KA task.

As well, a number of design enhancements to the 7Cs approach were suggested in Chapter 13 (page 242) and a number of implementation enhancements for FastFIX were suggested in Appendix O (commencing on page 451). It would be great if some of these suggested enhancements could be explored in future work.

In the past, there have been many hybrid KA and ML approaches, for example Gaines and Compton proposed the use of the *Induct* machine-learning algorithm (1995); Ware et al. provided a visual approach to support users in a combined KA and machine-learning approach (2000); McCreath and Kay proposed a hybrid KA and machine-learning approach for managing email (2003); and more recently, Bekmann (2006) combined KA with AI and ML techniques in the optimization of channel routes in VLSI design and separately in the optimization of traffic light controls (see section 4.4.1, page 60). Further to this, a possible area for further research is to use the 7Cs approach with ML-based autonomous agents in place of human users, in order to elicit semantics from text, for example on the web.

Remaining questions for the support centre are: What is the overall labour and human cost of codification in a MCRDR-based KBS? What are the financial and other benefits? Can the overall effort be justified? Obviously these questions can only be answered in the context of the target problem domain(s). It was beyond the scope of this thesis to perform a thorough evaluation of these questions for the HTG context. However as a result of the software trial of the proposed system, a very preliminary evaluation was made for the HTG dial-home problem domain and a summary was provided (section 12.5 on page 227).

Although much more time and data would be required to perform a thorough assessment of the questions arising from this research (in a related example, the PEIRS trial took four years), the results of the FastFIX software trial were most encouraging (section 12.14 on page 240). After less than a day of collective effort the test team had acquired enough knowledge for FastFIX to automatically identify and locate solutions to approximately 90% of problems in the selected sub-domain.

Therefore it is hoped that the research reported in this thesis will provide insights and benefits to the acquisition of knowledge in many different trouble-shooting, configuration, and classification domains in the future.

GLOSSARY

7Cs	A system that supports the Collaborative Classification and Configuration of a stream of incoming Cases via a relational structure of ConditionNodes, Classes and Conclusions (hence 7Cs)
Anchor Text	is the text that webmasters use to label their links to others' websites. For example in HTML the anchor reference: *Look At Me* applies the anchor text: *Look At Me* to refer to the website shown.
ANN	Artificial Neural Network
AI	Artificial Intelligence
Bottom Up	The term "bottom-up" knowledge acquisition refers to human users adding rules on the basis of a specific case on hand, in a manner that does not necessitate the user to peruse the rule tree, for example by allowing the user to create new RuleNodes that are relative to existing RuleNodes in the system.
A-V	Attribute-Value
C.A.R.D.	Case And Rule Driven
CBR	Case Based Reasoning
CD	Case-driven
CaseDB	HTG's case tracking tool
CORAL	Case Oriented Rule Acquisition Language
ConditionNode	see RuleNode
Cornerstone Case	A cornerstone case is one that first caused a given RuleNode to be created. From Compton and Jansen (1990): " The most important resource for knowledge maintenance is a data base of "cornerstone cases". These are cases which at some stage have required a change in the system's knowledge." From Richards (1998a, p55) "In single classification RDR only one case is associated with each rule. In

	MCRDR there may be multiple cases that must be distinguished from the current case." In this thesis, this latter set of cases is referred to as a "cornerstone case list". See also Table 19 on page 201.
Cornerstone Case List	In MCRDR systems, this is the set of cases comprised of the Cornerstone case for a given RuleNode, as well as the cornerstone case of each of its dependent RuleNodes. When the user wishes to reject the given RuleNode for a case on hand, the user will be shown the cornerstone case list for that parent RuleNode and invited to create a new child RuleNode that differentiates from all of the cases in its parent node's cornerstone case list.
CSI	Consortium for Service Innovation
DCL	Dependent Case List
DFD	Data Flow Diagram
DSS	Decision Support System
DM	Data Mining
DMQL	Data Mining Query Language
Docco	An alias for HTG's internal document management software system
ES	Expert System
FastFIX	The name of the prototype software developed in this thesis in order to trial the 7Cs design concept.
GDSS	Group Decision Support System
HCI	Human Computer Interaction
HSL	HTG's Hardware Support Lab
HTG	High Tech Global - an alias for the sponsor and target company
HWversion	An alias for the version number of the ProductA hardware
ICT	Information and Communications Technology
IR	Information Retrieval

I-RDR	Interactive RDR
IP	Intellectual Property
IS	Information Systems
KA	Knowledge Acquisition
KE	Knowledge Engineering
KBS	Knowledge Based System
KCS	Knowledge-Centred Support
KDD	Knowledge Discovery in Databases
KM	Knowledge Management
KPI	Key Performance Indicator
LCL	Live Case List – see section 11.3.1 (page 184)
LHS	Left Hand Side
LRL	Live RuleNode List see section 11.3.1 (page 184) and Table 14 (page 183)
MCRDR	Multiple Classification Ripple Down Rules
ML	Machine Learning
NN	Neural Network
N-RDR	Nested RDR
OLAP	On-line Analytical Processing
OO	Object Oriented
PG1	HTG's Product Group 1
PG2	HTG's Product Group 2
PKS	Pacific Knowledge Systems
SolutionDB	HTG's solution tracking tool
SWversion	An alias for the version number of HTG's ProductA software

ProductA	An alias for one of HTG's products
ProductB	An alias for another one of HTG's products
RCL	Registered Case List – see section 11.3.2 (page 185)
RHS	Right Hand Side
RRL	Registered RuleNode List – see section 11.3.2 (page 185) and Table 14 (page 183)
RD	Rule-driven
RDR	Ripple Down Rules
R-RDR	Recursive RDR
RuleNode	A node in a decision tree comprising of rule conditions, and optionally representing one or more classifications and conclusions. In the 7Cs system, a RuleNode is also referred to as a ConditionNode.
SCRDR	Single Classification Ripple Down Rules
SelfServeWeb	An alias for HTG's self-serve support website for customers
SolverDB	An alias for the solution tracking tool used by HTG's Engineering group.
Top-Down	The term "top-down" knowledge acquisition refers to the process of human users adding rules directly to a rule tree, without needing to refer to cases.
Trouble-shooting	Solving Problems i.e. eliminating trouble
TS1	HTG's Technical Support Level 1 Group
TS2	HTG's Technical Support Level 2 Group
URI	Uniform Resource Identifier

PUBLICATIONS

The following list shows the publications resulting from this research, listed chronologically from the earliest publication to the most recent. Assoc. Prof. Debbie Richards has been the academic supervisor at Macquarie University for this thesis.

1. Vazey, M. and Richards, D. (2004a) Achieving Rapid Knowledge Acquisition. Proceedings of the Pacific Knowledge Acquisition Workshop (PKAW), in conjunction with The Eighth Pacific Rim International Conference on Artificial Intelligence, August 9-13, 2004, Auckland, New Zealand, 74-86.
2. Richards, D. and Vazey, M. (2004) Managing Knowledge in ICT Organisations - A Solution for Corporate Support Centres. Proceedings of the Australian Conference for Knowledge Management & Intelligent Decision Support (ACKMIDS), Melbourne, Australia, 29-30 November 2004.
3. Vazey, M. and Richards, D. (2004b) Supporting Communities of Practice: A Ripple Down Rules Approach to Call Centre Management. Proceedings of the First International Conference on Knowledge Management (ICKM), 13-15 December 2004, Singapore.
4. Vazey, M. and Richards, D. (2005a) Trouble-shooting At The Call Centre: A Knowledge-Based Approach. Proceedings of the IASTED International Conference on Artificial Intelligence and Applications (AIA), February 14-16, Innsbruck, Austria.
5. Vazey, M. and Richards, D. (2005b) Intelligent Management of Call Centre Knowledge, Proceedings of the International Resource Management Association conference (IRMA), May 2005, San Diego, USA.
6. Richards, D. and Vazey, M. (2005) Closing the Gap Between Different Knowledge Sources and Types in the Call Centre. Proceedings of the Australian Conference in Information Systems (ACIS), November 2005, Sydney, Australia.
7. Vazey, M. and Richards, D. (2006a) A Case-Classification-Conclusion 3Cs Approach to Knowledge Acquisition - *Applying a Classification Logic Wiki to the Problem Solving Process*. International Journal of Knowledge Management (IJKM), Vol. 2, Issue 1, pp 72-88; article #ITJ3096, Jan-Mar 2006.

8. Vazey, M. and Richards, D. (2006b) Evaluation of the FastFIX Prototype 5Cs CARD System. Proceedings of the Pacific Knowledge Acquisition Workshop (PKAW 2006), August 7th and 8th, Guilin, China[193].

   Reproduced as "Evaluation of the FastFIX Prototype 5Cs CARD System" as a book chapter in the journal: "Advances in Knowledge Acquisition and Management" that is part of the Lecture Notes in Computer Science, Springer Berlin / Heidelberg, ISSN, 0302-9743, Volume 4303/2006, 10.1007/11961239, Copyright 2006, ISBN, 978-3-540-68955-3, 10.1007/11961239_10, Pages 108-119.

9. Vazey, M. (2006a) Stochastic Foundations for the Case-driven Acquisition of Classification Rules. Proceedings of the European Knowledge Acquisition Workshop (EKAW 2006), 2nd-6th October, Podebrady, Czech Republic[194].

   Reproduced as "Stochastic Foundations for the Case-Driven Acquisition of Classification Rules" in the journal: "Managing Knowledge in a World of Networks" that is part of the Lecture Notes in Computer Science, Springer Berlin / Heidelberg, ISSN, 0302-9743, Volume 4248/2006, 10.1007/11891451, Copyright 2006, ISBN, 978-3-540-46363-4, 10.1007/11891451_7, Pages 43-50.

10. Vazey, M. (2006b) Stochastic Foundations for the Case-driven Acquisition of Classification Knowledge in CBR systems. Proceedings of the 11th UK Workshop on Case-Based Reasoning, at the Twenty-sixth SGAI International Conference on Innovative Techniques and Applications of Artificial Intelligence (AI-2006).

[193] The acceptance rate for (Vazey and Richards, 2006b) i.e. for full papers at PKAW was 21 out of 78 i.e. 27%. The paper was blind reviewed by 3 reviewers.

[194] The acceptance rate for (Vazey, 2006a) at EKAW was 33 out of 119 submissions i.e. 28%. The paper was triple blind reviewed by 3 reviewers.

PATENT FILED

I created the system top level and detailed design, conceived of and wrote the complete detailed description, and drafted the plain English text of all the claims in U.S. Patent application PCT/AU2005/001087 (August, 2005). Access Macquarie contracted Spruson & Ferguson to provide appropriate legal wording and patent advice. Many thanks to Assoc. Prof. Debbie Richards for editorial support and review.

AWARDS

1. The Macquarie University Postgraduate Innovation Award, November 2005. This annual award is open to postgraduate students at Macquarie University as outlined at: http://www.mq.edu.au/innovationawards/criteria-postgraduate.htm. The award included a $AUD 2,000 cash prize and a bronze cast trophy.
2. The Dean's Prize for the best presentation in the research student seminar series in the Division of Information and Communication Sciences at Macquarie University during semester 2, 2006. The award included a $AUD 500 cash prize.