Achieving Rapid Knowledge Acquisition in a High-Volume Call Centre

Megan Vazey and Debbie Richards

Department of Computing
Division of Information and Communications Sciences
Macquarie University
Sydney, Australia
Email: megan@excelan.com.au, richards@ics.mq.edu.au

Abstract

Ripple Down Rules (RDR) has been applied to a number of domains. In this paper we consider a new application area that presents a number of new challenges. Our application is a high-volume call centre that provides a service / help desk function in a complex problem domain. We propose that the combined use of multiple classification ripple-down-rules (MCRDR) together with a web-enabled hyperlink-rich browser front-end will provide an effective tool to help call-centre knowledge workers cut through the potential information overload presented by both intra- and inter-nets; speed up the processes of knowledge acquisition and re-use; and assist with decision support and problem resolution.

We consider the implementation issues faced by corporations in their transition from a simple call / defect-tracking model to a much enriched knowledge-centered model and we examine the role MCRDR can play in the call-centre context including workflow integration, accessibility, usability, and incentives. In order to improve the fit with our application area, we suggest a number of variations to the MCRDR theme. Our implementation and evaluation of these ideas is ongoing.

1. Introduction

The Internet revolution of the mid 1990s has bought an unprecedented level of global knowledge and opinion to homes and offices alike. On both the academic and commercial front, it has prompted an enormous amount of interest and investment in knowledge management. The simple hyperlink has been a stunning force for change in the way we now perceive and work with knowledge. It has also created countless opportunities to track, review and comprehend the paths that users take through knowledge mazes presented by inter- and intra-nets alike.

One corporate sphere with an enormous thirst for knowledge is the customer call-centre. The last decade has seen globally explosive growth in call-centres providing both Customer Relationship Management (CRM) and help / service - desk functions: advising customers, answering queries, and resolving customer problems. In this domain, rapid access to appropriate, accurate and concise knowledge is paramount.

In this section we uncover the type of knowledge needed by call centre personnel to trouble-shoot vast volumes of sometimes simple, but often difficult,

technical problems in a dynamic knowledge environment. We propose an MCRDR expert systems approach and identify that challenges that this domain presents to the MCRDR paradigm. In section 2 we describe a number of previous MCRDR approaches to the Help Desk domain and how the scope of our project differs from these. In Section 3 we introduce the basic ideas behind MCRDR. Section 4 presents our solution, Interactive Recursive MCRDR (IR-MCRDR), describing the key issues and extensions that are needed to address the complex Call Centre environment. Our conclusions are given in the final section.

1.1 Call / Defect Tracking Software - Problem Ticketing

Historically, call-centers have placed call / defect tracking at the core of their query receipt and resolution process.

Typically, incoming calls are logged in a call / defect tracking database. Basic features of the incoming case are logged such as date, time, client name, and query summary. More specific details may also be included such as the name, model and / or version of any defective product (e.g. hardware or software) together with a query description.

Problem tickets may be machine generated e.g. with problem equipment ringing or emailing through the problem tickets. They may also be entered directly by customers, for example via a user-driven web interface. More traditionally they originate with front-line customer service personnel taking the first call.

The problem ticket passes through several states in which it moves between workers at various levels in the organization, for example from machine-generated, web-created, or front-line customer service personnel (state := *new*) to first or second tier customer service or technical support personnel (state := *assigned* then *opened* then *resolved*) then on to a team leader or even back to the customer (state := *closed*).

1.2 What Knowledge?

Somewhere between the *opened* state and the *resolved* state is where the real magic and art of problem solving by human experts is called upon. Here is where the customer service or technical support personnel go to their procedures manuals, training handouts, technical support home pages, knowledge databases, web search engines and inter / intra-nets to commence the often labor intensive process of finding out how to resolve the customer's query.

More experienced call-centre personnel create their own cheat-sheets, HTML link-rich home pages, and *uncovered* series to keep tabs on sources of knowledge likely to help them with their daily grind.

The problem, as we see it, is a perception of which knowledge is most relevant to the call-centre. And sometimes we can't see the wood for the trees…

In the call-centre context, one of the most important sources of knowledge is the knowledge of *how similar problems were solved in the past*. We believe that acquisition and re-use of this type of knowledge in the call-centre will deliver enormous benefit to customers and employees, and drastic bottom-line improvements for call-centers and their help / service desks.

Imagine, when a new problem ticket comes in, the customer service personnel is presented with a set of refinement queries enabling them to more specifically describe the type of problem being observed by the customer. Immediately that the new information is entered, the history of how *similar problems were solved in the past* is presented to the user – which internet links proved useful, and which knowledge-base references helped.

We believe that a simple extension to the problem-ticketing paradigm described above is all that is required.

1.3 An Expert System Approach

We believe expert system technology can be applied to record the decision making intelligence that call-centre and help / service desk personnel use to determine the resources (knowledge base, web, document or otherwise) that best assist with particular types of customer query.

We intend to augment the multiple-classification-ripple-down-rules (MCRDR) algorithm introduced by Kang, Compton and Preston (1995). MCRDR is a variation of the ripple-down-rules (RDR) algorithm developed by Compton and Jansen (1990) and described further in section 0.

We believe that the strengths of the MCRDR approach which includes easy knowledge acquisition, maintenance and validation performed directly by the domain expert, the customisability of the knowledge to suit the local environment, the use of cases to contextualise and assist knowledge acquisition are key reasons why MCRDR can offer a valuable solution in this area.

Closing the loop between the answers a user is searching for, and the quality of the answers retrieved is a vital step in training the system to excel in matching solutions to problems.

As time goes on, the cumulative effect of presenting more and more cases to our system is that the system gets trained and refined to achieve high levels of accuracy in matching solution resources to problem types. This will obviously be of huge benefit to the call center helpdesk - no more fumbling around with search engines, local web pages, or existing knowledge bases to find the relevant information. Experience with the MCRDR algorithm in other applications such as pathology (Edwards et al 1993) and the experiments of Kang (1996) suggest that such an expert system will grow rapidly in its level of matching accuracy as cases are added.

In our system, each MCRDR classification will be a set of one or more intra- or inter-net hyperlink references, where each reference points to web content that will assist with troubleshooting the current case.

MCRDR can be considered as a variant to the Case-Based Reasoning approach. CBR is appropriate where there is no formalized knowledge in the domain or where it is difficult for the expert to express their expertise in the format of rules (Kang et. al, 1996). However, MCRDR is more than just a case-based reasoner. MCRDR is also a rule based approach in that it uses rules to index the cases, thus addressing a problem associated with CBR systems that often require manual indexing. The cases motivate and assist rule development and the rules provide structure for storage and retrieval of the cases as we will see in section 3.

1.4 Challenges for MCRDR

The call-center help-desk context under consideration has a number of properties that present new challenges for the MCRDR algorithm:

- the system must interact with a legacy ticketing system and legacy knowledge base
- the system needs to deal with numerous cases (in the order of 50 per day locally, and 300 per day globally)
- the volume of cases being dealt with means that the workflow must inherently deal with system maintenance i.e. system maintenance must be in-circuit
- initial problem descriptions are sparse – the case definition matures as the customer service personnel interacts with the customer and *works the case*.
- while most cases are resolved promptly, a number of cases are open for days or even weeks
- problem receipt and resolution is asynchronous since there is a time delay (up to a day) between when the system receives a problem case, and when a customer service representative can attend to it.
- archived cases and the conclusions registered to them need to be available for several years (perhaps 10 years for some cases) into the future
- old cases may be edited
- multiple users will use and update the system, but a limited subset of privileged users will approve their updates
- old conclusions may be edited
- the granularity of conclusions may vary widely and conclusions that are web links may expire

Further this project is concerned with solving various call center problems using MCRDR for a real organization which imposes further characteristics on the problem and constraints on the solution.

- There are very many attributes which will vary across cases and new attributes will frequently need to be added.
- The range of values possible for those attributes is also very large and the dependencies between these A-V pairs may be very strong. For example, we are dealing with troubleshooting across multiple systems, platforms, vendors, versions, etc.
- We don't have control over the cases, which are stored in the parent company's database.

2. MCRDR-based Help Desk Approaches

MCRDR has been explored in the help desk environment by Kang, Yoshida, Motoda, Compton in 1996; Kim, Compton and Kang in 1999; and again by Kim in 2003. In the subsections that follow we describe their work and, in the final subsection, we note the key differences in comparison with our own work.

2.1 A Help Desk System with Intelligent Interface

The prototype described by Kang, Yoshida, Motoda, Compton (1996) combined a keyword search with Case-Based Reasoning indexing techniques to provide a guided MCRDR interaction that was able to quickly steer users to appropriate help information on the internet. Their system considered updates by a single expert only.

As noted by Kang et. al. (1996) the MCRDR engine has two problems as an information retrieval engine. The first one is the number of conditions that are to be reviewed by the user. The second one is the number of interactions between the user and the system.

Their prototype attempted to minimise this problem by allowing users to apply a keyword search to effectively pre-filter the rule tree to only include those cases that satisfied the keyword search criteria. The user could then interact with a minimised MCRDR rule tree to select the relevant cases and update the knowledge accordingly.

The idea is compelling and may prove to be a useful adjunct for browsing the knowledge in our system.

2.2 Incremental Development of a Web Based Help Desk System

The system described in section 2.1 implemented an information retrieval function for users. The prototype described by Kim, Compton, and Kang (1999) extended this prototype by allowing an expert user to also build and maintain the help desk document knowledge base by applying keywords to help documents.

2.3 Document Management and Retrieval for Specialised Domains: An Evolutionary User-Based Approach.

In her PhD thesis, Kim (2003) re-evaluated the prototype described in section 2.2 and applied the concept lattice from Formal Concept Analysis (FCA) (Wille 1992) to generate a browsing structure to assist users in navigating the knowledge base.

2.4 A Very Different Help-Desk Application

Our system differs considerably in scope from the previous RDR help-desk systems described above, in particular:

- The A-V pairs and rules in our system are not simple keywords, and simple tests for existence of keywords. Rather, the attributes may be any type e.g. integer, float, string, enumerated type, or free-text; they may be single valued, one of a set, or some of a set; and tests may include tests for range such as ‘installation date > 2001/01/30’; for existence (indicated as ?) such as ‘? patch 3.6.5’; for containment e.g. ‘case description contains *machine generated*’ or for equivalence e.g. ‘version == 3.2’.
- Multiple users will describe the cases through an interactive question-answer interface to the system that will assign the relevant A-V pairs to the case.
- Our system will be maintained by multiple users, not just a single user.
- Knowledge acquisition and system maintenance needs to be in-circuit – with

50 cases per day to handle locally, users won't wait for a knowledge engineer to *get back to them.*

- Our system needs to fit smoothly into the workflow of a bustling call centre – expediency, efficiency and accuracy will be key to the system's success.

3. Introducing Multiple Classification Ripple Down Rules

The MCRDR algorithm (Kang, Compton and Preston, 1995) is now a decade old and in that time it has seen numerous implementations. In a programmatic sense, the algorithm is clearly and concisely explained in its founding paper, however, we include our own brief description of the MCRDR decision tree in Table 1 below. An example MCRDR decision tree is provided in Figure 1.

Table 1: Description of the MCRDR Decision Tree

1.	There is an N-ary Tree of RuleNodes.
2.	Each RuleNode has a *rule* and a *conclusion*.
3.	The topmost RuleNode in the tree evaluates to TRUE for every case in the system.
4.	Cases comprise of *attribute-value* pairs eg in the case of a frog, *'voice'* == *'croaks*, *'movement'* == *'hops'*.
5.	The *rule* at each RuleNode tests the attribute properties eg that *'movement'* == *'flies'*.
6.	Each case is evaluated against the topmost parent RuleNode and then successively down the tree for each child RuleNode.
7.	If the result is TRUE for a parent Rulenode, the case is recursively evaluated against all of its child RuleNodes.
8.	The conclusions for a Case are given by the last TRUE RuleNode in every path down the RuleNode Tree.

As an example, in the case of a bird in Figure 1 where ('movement' == 'flies') and ('voice' == 'sings'), the last TRUE RuleNode in every path down the RuleNode Tree gives the conclusions 'sweet sounding', 'bird'.

Note that the symbols LRL, RRL, LCL, LPL, and RCL in Figure 1 are unique to our implementation. They stand for Live RuleNode List, Registered RuleNode List, Live Case List, Live Path List, and Registered Case List. These concepts and are explained in section 4.7.

When a new case is added to the system, the user can choose to accept a given conclusion or alternatively reject it by creating a differentiating rule with an alternate conclusion. In that case:

- The new rule must be a valid boolean expression which is able to be evaluated by the MCRDR engine.The rule for the new RuleNode should be different from the rules of its ancestor RuleNodes.
- The rule for the new RuleNode may optionally be restricted to a single test

eg that ('movement' == 'flies'), rather than a conjunction of tests[1].

- The new RuleNode must have either a different conclusion, or a different rule compared to it's sibling RuleNodes.
- The new RuleNode must test for some feature of the Review Case and must evaluate to TRUE for the Review Case.
- The new RuleNode must distinguish between the Review Case and all of the Cornerstone Cases for the parent RuleNode.

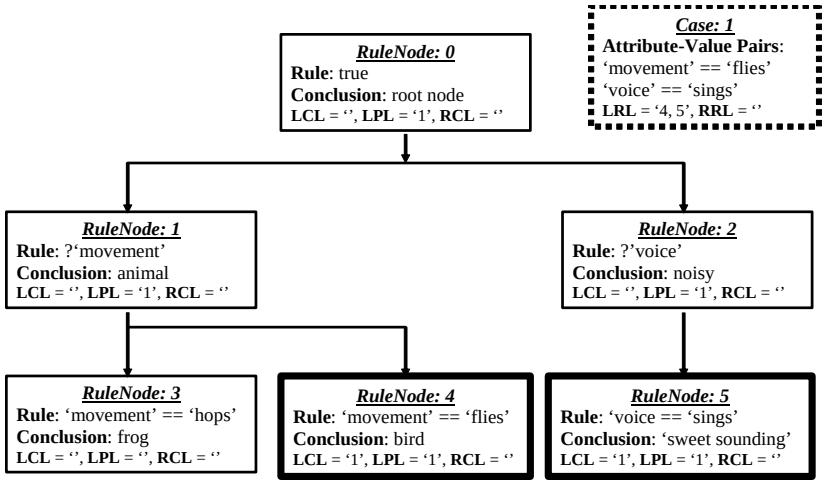

Figure 1: MCRDR Decision Tree (Case == Bird)

New RuleNodes can generally[2] be placed at one of two places in the tree (Kang, Compton and Preston, 1995):

- At the top of the RuleTree to provide a new independent conclusion.
- Beneath the current RuleNode as a replacement conclusion or as a stopping conclusion.

Further details regarding our implementation are provided in section 4.

4. A Solution for Call-Centre Service-Desks

Keeping in mind the challenges for MCRDR described in section 1.4, we intend to explore a number of variations to the MCRDR theme. These variations are described in the subsections that follow.

Figure 2 presents a condensed entity relationship (ER) diagram that

[1] Where more complicated conjunctions of tests are allowed, the new RuleNode is more likely to be added to the top of the tree and a stopping rule used at the end of the path – the overall result is a flatter rule-tree structure (Kim, 2003).

[2] Actually, there is a possibility that new RuleNodes could be placed in the path between the topmost RuleNode and the current RuleNode by asking the user to identify the minimum set of rules in the current path that the case must satisfy for the new RuleNode.

summarises the relationships between data structures used in our system. Unfortunately, space does not permit us to include a detailed ER diagram. However, in this section, we provide a detailed description of the structure.

Referring to Figure 2, Cases contain a history of case statements that have been added to the case over time by users.

Cases also contain a list of Attribute-Value (A-V) pairs where each attribute has a name; a type (for example: one-of-a-set, some-of-a-set, float-with-range, integer-without-range, or free-text); a set of accepted-values (as required by one-of-a-set, some-of-a-set or ranged attribute types), the attribute display units (for example kilograms, metres); the order in which the attribute should be displayed relative to other attributes; and the System Log or History showing who created the attribute, who has modified it, and when these events occurred.

A key aspect of our system is that the structure provides for a many to many relationship between Cases and RuleNodes via the conceptual use of the Registered RuleNode List, Approved RuleNode List, and Live RuleNode List stored with each case; and via the Registered Case List, Approved Case List, Live Case List, and Live Path List stored with each RuleNode. The purpose and importance of this unique structure is explained in detail in section 4.7.

A RuleNode Assoc(iative) List in the Case Table, and a Case Assoc(iative) List in the RuleNode Table, together with the Case-RuleNode Association Table, provide an alternate implementation that is designed to allow for more effective management of the change history of the associations between Cases and RuleNodes.

In keeping with the MCRDR decision tree described in section 3, each RuleNode contains a reference to its parent RuleNode, its child RuleNode if one exists, and its sibling RuleNode if one exists. RuleNodes also contain a record of their cornerstone cases (which must be both live and registered).

RuleNodes contain a Rule Statement, which is a boolean expression that can be evaluated by the MCRDR engine to determine the truth of that rule for a given case.

RuleNodes also contain a set of Conclusion Statements where each conclusion statement can be an internet URL, plain text, or an instruction to the MCRDR engine to interactively prompt the user for more A-V details and then recursively re-evaluate the case. Our design provides for multiple RuleNodes to refer to a given Conclusion Statement.

We provide RuleNode and Conclusion confidence scores by analysing the usefulness rating (0 to 5) assigned by users (this mechanism may be augmented to provide a confidence in the context of a given RuleNode).

Cases, RuleNodes, Attributes, and the system itself can be commented on and those comments can be organized into internet forum-style threads.

Users are given a username and password, their job role is identified, and counts are kept of the number of cases that they have augmented, the number of cases they have closed, and the number of RuleNodes that they have created. Together with an indication of how long they've been using the system, these counts are used to assign users with an overall credibility score.

One idea is to link staff incentives to the User Credibility Score, for example by giving out movie tickets for "gold users" – those that provide the most used

and highest rating conclusions and / or RuleNodes.

Our design provides for the possibility that Cases, RuleNodes, Rule Statements, Conclusions, Case-RuleNode Associations, and Attributes may be all the subject of asynchronous editing. It remains to be seen whether such an approach will overwhelm users with choice. In that case, we may scale back some of the flexibility to restricted use by the system's Administrators.

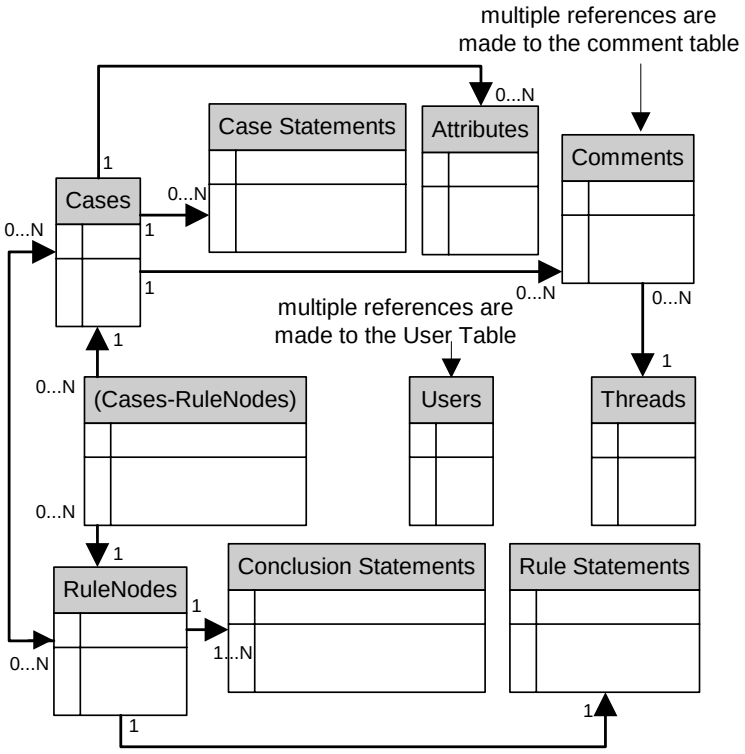

Figure 2: Enhanced (Condensed) MCRDR Entity Relationship Diagram

4.1 IR-MCRDR

Expediency, accuracy, and efficiency are key performance criteria for the call-centre. This means that our solution needs to be well integrated with the natural workflow of the call-centre, and in particular, it must be designed for minimal user-decisions.

One problem presented by existing defect tracking and knowledge management solutions is that users are presented with long lists of A-V pairs, many of which are irrelevant to the problem on-hand. It is left to the user to apply a mental filter for each new case when filling in these A-V pairs.

Importantly, we aim to reduce the decision burden for users of the system, and

thereby speed up the process of problem determination as well as reduce the risk of information overload to the user. In this endeavor, we intend to apply and extend two variations to the RDR theme: Recursive RRDR (Mulholland et al 1993) that involved repeated inference cycles using the single classification RDR structure; and Interactive RDR (IRDR) which was a technique that allowed the RDR system to prompt the user for more information when required. Hence we propose the development of IR-MCRDR.

Our idea is to use IR-MCRDR in a configuration sense. Our system will assist the user in honing the problem definition by using the current case context to prompt the user for more detail about the specific problem being considered. Only relevant A-V entries will be requested of the user, depending on the current case context.

In addition, as more A-V pairs are gathered to define the case, the case will invoke conclusions that lie deeper down the rule paths of the decision tree, and the conclusions displayed will become more specific to the particular problem being observed.

Our hope is that this approach will quickly guide users to the most relevant conclusions for the problem case under consideration.

4.2 Relaxing the Case Differentiation Test for new RuleNodes

With previous implementations of MCRDR, a new RuleNode can only be added when the new rule differentiates the case in question from all the cornerstone cases at the node above. This means that when a new RuleNode is added, all of the cases that previously tested TRUE for the parent node are now evaluated at the new child node.

In our system we must retain references to old cases. Therefore, there could be hundreds, if not thousands of cases at that node.

We propose that the user need only optionally differentiate against a limited number of cases at the parent RuleNode, namely those that are both *live* and *registered*, that present a unique set of A-V pairs, and possibly those that meet a certain expiration threshold, for example less than 1 year old.

Relaxing the case differentiation test raises issues of knowledge consistency and version control. The same type of problem can occur with rule or case edits in an MCRDR system. Suryanto explored this issue in his thesis when he compared NRDR and Repeat Inference MCRDR (RI-MCRDR). With traditional MCRDR, even if a RuleNode is modified in the RuleTree in a manner that would fetch a new conclusion for an old case (if it were re-evaluated against the revised rule tree), the classification for the old case will never get updated.

In NRDR (Beydoun and Hoffman, 1997), abstracted RDRs are able to be edited in a manner that can fetch new conclusions for old cases. In that instance, the expert is required to maintain knowledge consistency by providing secondary refinements for the affected case.

According to Suryanto (personal communication - June 2004), our approach therefore lies between NRDR and RIMCRDR:

- NRDR requires secondary refinement for affected cases,
- In our approach, the secondary refinement of affected cases is delayed until such time that it suits the user to make that refinement, and
- In MCRDR, the secondary refinement of affected cases is never required.

The central issue is therefore the benefit and cost of the secondary refinement.

4.3 Separation of Live and Registered RuleNodes (Conclusions)

We keep a record per case of *live* vs *registered conclusions* i.e RuleNodes. Live conclusions are those that are current given the present iteration of the rule tree. Registered RuleNodes are those that were once agreed to by a user, but which may now appear old relative to the current knowledge kept in the rule tree.

Maintaining a list of live vs registered Rulenodes means that a record of how conclusions for an individual case have changed over time is not lost by the system. Users can query the system to find all the cases that have "dropped through" from one conclusion to the next for one reason or another (eg NRDR definitional change, case edit, rule edit, rule addition that wasn't tested against this case because the number of cases were too numerous for the user to consider).

Additionally, users don't need to correct the conclusions for legacy cases unless they specifically want to. Cases are allowed to hang around in the system in a state where their live conclusion list (ie the list of current system conclusions for the case) is different to their registered conclusion list (ie the list of conclusions that were true for the case last time someone synchronised that case with the knowledge base).

This mechanism guards against the undesirable scenario where a user may examine a case that they dealt with in the past, only to find that the system generated (live) conclusions have changed compared to the conclusions that the user themselves registered for that case, or perhaps worse in a dynamic knowledge environment, the scenario where old cases are never updated with new and knowledge and hence new conclusions.

This proposal gives rise to the decision scenarios shown in Table 2. For example, the system evaluates an old case and shows the user that there is a registered RuleNode (conclusion set) that is no longer live – perhaps the case has been edited or the case now falls down to a new RuleNode (conclusion set) – the user can either reject the conclusion (010) to *Remove this RuleNode from the Registered List*, or accept the conclusion (011) to *Create a New RuleNode to Accept this Conclusion*.

Table 2: Decision Scenarios, a Truth-Table

Live RuleNode (Conclusion)	Registered RuleNode (Conclusion)	Accept (1) or Reject (0)	Action
0	0	0	Do nothing
0	0	1	Create a New Rulenode and Conclusion
0	1	0	Remove RuleNode from the Registered List
0	1	1	Create New RuleNode to Accept this Conclusion
1	0	0	Create New RuleNode to Reject this Conclusion
1	0	1	Register this RuleNode
1	1	0	Create New RuleNode to Reject this Conclusion
1	1	1	Do nothing

4.4 Managing Approvals

Obviously it is vital to the success of our tool that it actually gets used – *the more it is used, the more useful it will prove to be.* In this vein, our implementation will allow multiple users to dynamically update the knowledge.

User's confidence in the presented solutions will also play a large part in users wanting to use the system. We intend to provide for an approval mechanism that allows one or more trusted experts to indicate their approval of a given solution.

We propose a third RuleNode state entitled *approved RuleNode*. An *approved RuleNode* is one that was initially *live*, was subsequently *registered* by a user, and finally *approved* by a trusted expert for a given case. If further down the track the case is edited, or the rule tree modified in such a way that the *live RuleNodes* no longer match the *registered RuleNodes*, then that case may be subjected to *re-registration* and *re-approval*.

Table 3 shows a proposed presentation format for the live, registered and approved RuleNode states, and the user options to accept or reject each conclusion.

Table 3: Live, Registered, and Approved Conclusion States for Case 1: A Sweet Sounding Bird

RuleNode Details						Recommendations	
Rule Node	Rule	Conclusion Set	Live	Registered	Approved	Accept	Reject
4	'movement'=='flies'	bird	yes	meganv 2004/03/30	richards 2004/05/06	◉	○
5	'voice'=='sings'	sweet sounding	yes	meganv 2004/04/01	-	○	◉

4.5 Recording a Change History per Case and per RuleNode

Given that old and expired cases may *fall through* to new conclusions, or that old cases or even old conclusions may be edited and modified, a system log or

change history can be kept per case showing how the case has changed and how its conclusion list has evolved over time.

Similarly, given that conclusions containing web references may expire and require update, a change history per Conclusion and per RuleNode is also required.

4.6 Building Credibility

We intend to satisfy user-demand for credibility by presenting interested users with a record at each RuleNode of how many cases presently refer to that RuleNode for conclusions that are both *live* and *registered*.

For each RuleNode, the system will record (in the System Log) the case that first caused the RuleNode to be created, and all the other cases that have at one time referred to it.

Users typically wouldn't need to know about RuleNodes at all. They would simply see the information presented as approved conclusions together with a credibility statement for each conclusion.

For each RuleNode, and hence conclusion, a trace of the path through the rule tree is also provided. We see this as fulfilling the role of an explanation of worthiness for system generated conclusions. As discussed by Doyle, Tsymbal, and Cunningham (2003) the use of explanations increases user acceptance of the predictions offered by knowledge based systems. They help to justify predictions in complicated domains where the domain is not fully understood or complicated heuristics are used.

The registered, live and approved case lists record the manner in which existing cases are presently linked to this RuleNode. Note that the Live Path Case list is explained in the next section.

4.7 Optimising the Rule Tree to Handle Numerous Cases

Given the large number of cases that need to be handled, the underlying data structure is critical. In our implementation, the following optimised tree structure is used:

- RuleNodes are evaluated on a once-only as-needs basis for each case (unless the case or the rule statement is modified). Most of this processing is done in the background when the case is first received. The result is recorded in the rule tree.
- Conceptually, the live N to N relationship between RuleNodes and Cases is recorded via the 'Live Case List' (LCL in Figure 1) at each RuleNode and via the 'Live RuleNode List' (LRL in Figure 1) within each Case.
- When a new RuleNode is added, the system only evaluates cases that are TRUE for the immediate parent node - the 'Live Path list' (LPL in Figure 1) at each RuleNode records these cases. The net result is that each iteration of each Rule Statement is only evaluated for cases in its referring RuleNode's 'Live Path List' (LPL). This saves significantly in processing overhead by reducing the number of cases examined. Note that LCL is a subset of the LPL at a given RuleNode.
- The user can re-open old cases. In that instance, the case is re-evaluated

from the top of the Rule Tree down. Note that each iteration of each case is only evaluated by the MCRDR engine against the Rule Tree once.

- As discussed earlier, conceptually the 'Registered Case List' (RCL in Figure 1) and 'Registered RuleNode List' (LRL in Figure 1) records the relationships between a case and its registered RuleNodes.

Whenever a new RuleNode is added, or a Case, or Rule Statement is edited, the system updates its internal references to maintain database integrity.

4.8 Legacy Problem Ticketing System and Knowledge Base

Call-centres have made and continue to make enormous investments in evaluating and purchasing workflow software. Quite separate from the financial investment in software is the investment in organizational learning - "*the way we do things around here*". This extends beyond the training of front-line personnel to an investment in custom reports and metrics to assist in performance management of the call-centre.

We intend to develop a MCRDR-based expert system that can operate independently and that can be used to augment any legacy ticketing or workflow system that the call-centre may have already invested in.

Our system will rely heavily on hyperlinks to existing intranet databases to achieve this end.

5. Conclusions

There are a number of other issues we have considered concerning feedback and collaboration, managing incentives to users to encourage the entry of good conclusions, the handling of conclusion granularity and expiration, accessibility to the system globally and continuously, usability and the possible inclusion of data mining to automatically extract knowledge from previously recorded decision data. We do not have space here to discuss our proposed strategies to these issues but acknowledge that the success of the system will only be possible if they are adequately addressed.

We have presented in this paper an overview of the call-centre domain and outlined some of the challenges it raises. We propose the use of a back-end MCRDR knowledge base supported by a web browser front-end to assist knowledge workers solve the many problem reports they receive daily. Due to the features of this domain, particularly the evolving nature of the cases themselves, we have suggested a number of modifications to standard MCRDR and have proposed Interactive Recursive MCRDR.

On the implementation side a prototype has already been developed. The key obstacles we currently face are access to the existing databases containing problem cases and solutions (in separate databases and formats) and having to develop a system that will not only interface with current systems but fit into the corporate structure and culture. These softer issues have always been the real challenges in knowledge acquisition.

Acknowledgement.

This work has been generously funded via the Australian Research Council Linkage Grant LP0347995. We wish to thank our industry partner for their support and assistance.

References

Acorn, T. and Walden, S. (1992) SMART: Support Management Automated Reasoning Technology for Compaq Customer Service. In Proc. of the 4th Innovative Application., of Artificial Intelligence Conference. 1992.

Bareiss E. R. (1989) Exemplar-Based Knowledge Acquisition: A Unified Approach to Concept, Representation, Classification and Learning Academic Press, Boston.

Beydoun, G. and Hoffman, A. (1997) NRDR for the Acquisition of Search Knowledge. Proceedings of The Tenth Australian Conference on Artificial Intelligence, Perth Australia, 1997, Springer.

Compton, P. J. and R. Jansen (1989). A philosophical basis for knowledge acquisition. 3rd European Knowledge Acquisition for Knowledge-Based Systems Workshop, Paris: 75-89.

Doyle, D., Tsymbal, A. and Cunningham, P. (2003). A Review of Explanation and Explanation in Case-Based Reasoning. D2003, TCD CS report.http://www.cs.tcd.ie/publications/tech-reports/reports.03/TCD-CS-2003-41.pdf]

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

Kang, B., P. Compton and P. Preston (1995). Multiple Classification Ripple Down Rules : Evaluation and Possibilities. Proceedings of the 9th AAAI-Sponsored Banff Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff, Canada, University of Calgary.

Kang, B., Yoshida, K., Motoda, H., Compton, P. and Iwayama, M., (1996) A help desk system with intelligent interface in P Compton, R Mizoguchi, H Motoda & T Menzies, PKAW'96: Pacific Rim Knowledge Acquisition Workshop, 1996 (Sydney, Dept. of Artificial Intelligence, School of Computer Science and Engineering, UNSW, 1996), pp 313-332.

Kim, M., Compton, P., Kang, B. (1999) Incremental development of a web-based help desk system. Proceedings of the 4th Australian Knowledge Acquisition Workshop (AKAW 99), University of NSW, Sydney, 5-6 Dec 1999, 13-29.

Kim M. (2003) PhD thesis, Document Management and Retrieval for Specialised Domains: An Evolutionary User-Based Approach. UNSW, 2003.

Kriegsman, M. and Barletta., R. (1993) Building a Case-Based Help Desk Application, IEEE Expert Vol 8(6) December 1993: 18-26.

Mulholland, M., Preston, P., Sammut, C., Hilbert, B. and Compton, P. (1993) An Expert System for Ion Chromatography Developed using Machine Learning and Knowledge in Context *Proc. of the 6th Int. Conf. On Industrial and Engineering Applications of Artificial Intelligence and Expert Systems* Edinburgh.

Simoudis, E. (1992) Using case-based reasoning for customer technical support IEEE Expert 7(5): 7-13.

Shimazu, H., Shibata, A. and Nihei, K. (1994) Case-Based Retrieval Interface Adapted to Customer-Initiated Dialogues in Help Desk Operations. AAAI 1994: 513-518

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