CHAPTER 11: THE 7CS AND FASTFIX DESIGN CORE

11.1 Chapter Outline

The 7Cs system was created to help trouble-shooters in any problem domain repetitively and collaboratively solve their information retrieval (configuration and / or classification–based) problems. FastFIX provided a prototype implementation of the 7Cs design. This chapter provides definitions together with screenshots of the FastFIX prototype, with the aim of providing an overview of the main concepts and features in the 7Cs system.

The web-based application shell of the FastFIX prototype is presented in Appendix M (page 424), including the User Security model and the FastFIX prototype GUI. Once the dial-home problem domain at HTG was selected, some customisation of the FastFIX graphical user interface (GUI) was made to simplify and streamline the trouble-shooting experience for users. These changes were minor and amounted to less than a week of customisation. Note that the core features discussed in this chapter could be implemented in many different conceivable shells, not just the shell afforded by the FastFIX prototype.

At this point, it may be helpful to draw the reader's attention to some of the key philosophical differences between the 7Cs approach and conventional MCRDR:

1. The prevailing view in conventional MCRDR is that knowledge bases only exist for cases, so KA should only ever occur using cases. This was highlighted previously in sections 8.3.1 (page 140), 8.3.6 (page 148), 8.3.10 (page 155) and 8.4 (page 158). In contrast, the 7Cs philosophy is that knowledge bases only exist for cases, but KA from human users can be achieved by acquiring either rules or cases. Requiring the user to always put their rules in the context of cases can be cumbersome in human-computer interaction (HCI) terms. The system can be tested with cases, but not every RuleNode in the system needs a case associated with it. As discussed previously in section 8.3.7 (page 150), this is particularly true for characteristic RuleNodes. Accordingly for ease of the 7Cs KA system has been designed to handle both top-down rule-driven and bottom-up case-driven KA.

2. As discussed previously in sections 8.3.1 (page 140), 8.3.8 (page 151), 8.3.9 (page 153), and 8.4 (page 158), in conventional MCRDR approaches it has been assumed that experts make few errors in entering the knowledge. However in the support

centre domain, many of the users are novices, and much of the knowledge is uncertain, so users can make many errors. From point 1 above, the 7Cs philosophy is that both top-down and bottom-up KA mechanisms are needed to support users in correcting errors in the knowledge base.

3. In conventional MCRDR and N-RDR approaches it is assumed that users already have the knowledge that needs to be acquired so they can enter it as soon as the need arises. However in the support centre domain much of the knowledge is dynamic and evolving so there can be significant delays between the user realising that some knowledge is missing in the knowledge base, and the user finding out what knowledge to enter and how best to enter it. The 7Cs philosophy is that the system needs to support a delay in users entering knowledge to the knowledge base, so that users can do this when the knowledge becomes available, and at a time that best suits them.
4. In conventional MCRDR approaches it is assumed that only one expert need update the system at a time. In contrast, the 7Cs philosophy is that knowledge will be more rapidly acquired and higher quality knowledge will result from allowing multiple experts to contribute to and negotiate within a shared knowledge acquisition framework.
5. Knowledge acquisition by definition involves human users, with human error, conflict and dynamic knowledge. Hence the 7Cs philosophy is that addressing human-computer-interaction (HCI) issues like flexible user input facilities and minimal wait-times is incredibly important in the KA implementation. Attention must be given to optimising the underlying data structures. External simplicity for the user may come at the (acceptable) cost of increased internal software complexity.

Further to these key philosophical differences, significant novel ideas included in the 7Cs system, implemented in the FastFIX prototype, and tested during the FastFIX software trial include:

- The ability for multiple users to build an MCRDR-based decision tree in a wiki-style collaborative effort. This includes the identification of classes of incoming problem cases and manual indexing of solutions by multiple users using rule conditions analogous to logical tags in a folksomony.

- The separation between the *live* case-RuleNode associations that represent the current global truth calculated and recorded by the 7Cs expert system, and the *registered* case-RuleNode associations that represent the current truths of individual contributors to the expert system.
- The ability for users to "work-up" a case using a novel Interactive and Recursive MCRDR decision structure.
- The ability for users to edit previously created cases (including cornerstone cases) and RuleNodes in the system.
- Continuous background monitoring of changes to the knowledge base so that users with affected ConditionNodes and Cases can notice and respond to the changes. This approach allows classification conflicts to be identified, clarified and resolved and hence it enhances knowledge acquisition.
- Separation between classifications and conclusions so that richer classification relationships can be maintained.
- Reference to multiple exemplar cornerstone cases for each ConditionNode.
- The ability for users to relocate i.e. move ConditionNodes in the system.

The creation, formation, development and testing of these novel ideas with real users in a real trouble-shooting environment has been one of the most significant contributions of this thesis.

The chapter has been organised into features pertaining to:

- Users,
- ConditionNodes and the Condition Mesh (Conditions, Classifications and Conclusions),
- Attributes,
- Cases,
- Change Histories and
- Case-ConditionNode Associations

Appendix M (page 424) describes additional features implemented in the FastFIX prototype. Design enhancements for future embodiments of the 7Cs system are proposed in Chapter 13 (page 242).

11.2 Users

As shown in Appendix M.2 (page 426), users are identified via a username and password so that their activity can be tracked in the system. A super-user can modify a user's "user type" to modify that user's privileges. In the future, it may be useful to allow users to not just be human users, but also computer users such as intelligent agents, or computer robots.

11.2.1 "My Details"

The following screenshot show the details currently held for a user in the FastFIX prototype.

Figure 24: My User Details

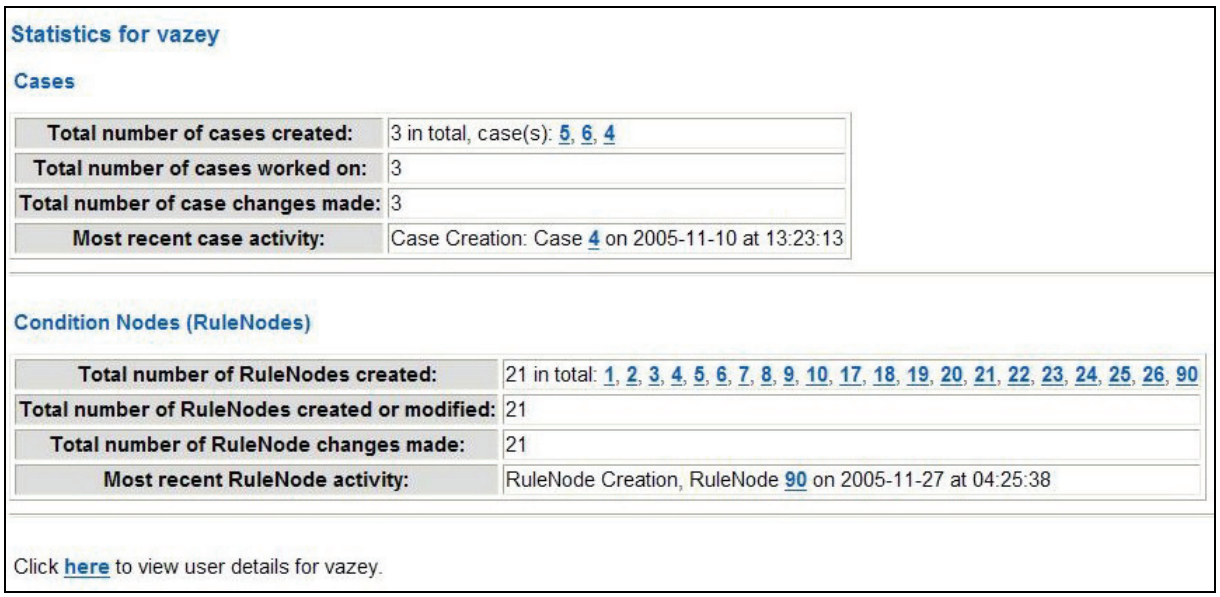
11.2.2 "My Statistics"

The idea of using incrementally accrued, context-specific performance statistics to enhance the credibility of the reported conclusions was introduced to SCRDR by Edwards (1996, p i, vi, 208). Performance statistics are also reportedly recorded by PKS's LabWizard product (see section G.12, page 411). However, since conventional RDR systems allowed only one expert at a time to add knowledge to the KBS, no user-based statistics were kept in relation to those users who modified the KBS.

In contrast, in the 7Cs system experts are encouraged to assess each other's credibility by publishing user-based knowledge acquisition statistics such as the number of times an expert's RuleNodes are used, corrected, augmented, approved, or overturned; and the number of times their cases are revisited and their previous decisions corrected.

The more that users use the system, the more refined the knowledge in the system will become, and the more useful the system will be. Therefore, the 7Cs system aims to capture statistics that will help users obtain confidence in the information presented. As shown in the next figure, statistics are accessible for every user in the system showing the number of cases and RuleNodes that each user creates, works on or makes changes to.

Figure 25: My User Statistics



Statistics for vazey

Cases

Total number of cases created:	3 in total, case(s): 5, 6, 4
Total number of cases worked on:	3
Total number of case changes made:	3
Most recent case activity:	Case Creation: Case 4 on 2005-11-10 at 13:23:13

Condition Nodes (RuleNodes)

Total number of RuleNodes created:	21 in total: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 90
Total number of RuleNodes created or modified:	21
Total number of RuleNode changes made:	21
Most recent RuleNode activity:	RuleNode Creation, RuleNode 90 on 2005-11-27 at 04:25:38

Click here to view user details for vazey.

Apart from the intrinsic motivation to use the system that making this information public may provide for some users, it may be possible to encourage users to increase their usage and hence their User Credibility Scores with extrinsic individual and / or team motivators, for example by giving out movie tickets or t-shirts for "gold users" – those that provide the most used and highest rating RuleNodes.

Note that in Figure 25 the hyperlinks for cases and RuleNodes are underlined and can be clicked on by users to bring up the corresponding case or RuleNode views.

The following table shows some of the statistics that could be recorded in future system embodiments.

Table 13: User statistics

Item	Statistics kept per user
Attributes	Number of attributes created; Number of attributes edited; Most recent attribute activity.
Cases	Total number of cases created; Total number of cases worked on; Total number of case changes made; Most recent case activity. Total number of times a user's cases have been revisited and their previous decisions corrected. Total number of cases assigned to this user, opened by this user, resolved by this user, verified by this user, closed by this user. Total number of cases currently being tracked for this user. Total number of this user's tracked cases that are currently dropped-through. Total number of case drop-throughs[169] for this user since the commencement of the system.
RuleNodes	Total number of RuleNodes created; Total number of RuleNodes worked on; Total number of RuleNode changes made; Most recent RuleNode activity. Total number of intermediate RuleNodes created; Total number of shared child RuleNodes created; Total number of RuleNodes with classification labels created. Total number of RuleNodes created that have been stopped. Total number of RuleNodes created that have been approved across each approval partition, and across all approval partitions. Total number of times a user's RuleNodes have been used, corrected, augmented, approved, or overturned.
Conclusions	Number of conclusions created; Number of conclusions edited; Most recent conclusion activity. Total number of hyperlink / getAttribute / setAttribute / and stopping conclusions created.

Note that in his evaluation of PEIRS, Edwards suggested that conventional RDR implementations lent themselves to credentialing via statistics since RuleNodes couldn't be modified or deleted. He argued that for other rule-editable expert systems, context-specific performance indices would not be possible (Edwards, 1996 p210). However, allowing RuleNodes to be edited and moved need not compromise the ability of the system to maintain statistics and hence credentials. In the proposed 7Cs system, a change history is kept for all elements of the system, including case-RuleNode associations. Statistics can be suitably recorded and reported according to each phase in the history of changes to the RuleNode. For example: "*While the RuleNode said A and was located under X, N cases were registered*

[169] Case drop-through was described previously in 6.5 on page 84.

against it. In its current location and form, M cases have been recorded against it. In total N + M cases have been registered against it." Each of these line items in the credentials history could be selected for inclusion or exclusion from the summary of credentials reported for the RuleNode.

11.2.3 "View User Stats"

The next figure shows a summary for all users in the system. From this figure we can see that during the trial of the FastFIX prototype, 12 users registered themselves in the system, 172 Cases were created, and 107 RuleNodes were acquired.

Figure 26: Statistics for all users in the system

View User Statistics

User ID	Username	First name	Second Name	Email	Website	User Level	Number of Cases Created	Number of RuleNodes Created
5						silver	64	42
9						wood	0	0
6				(Commercial in confidence)		stone	15	13
3						wood	2	0
8						wood	2	0
1						iron	83	30
12						wood	1	1
4						wood	1	0
7						wood	0	0
11						wood	0	0
10						wood	1	0
2						bronze	3	21
						TOTAL	172	107

Referring to the "User Level" in the above figure, a very simple credibility model was implemented in the FastFIX prototype using the key shown in Figure 27.

Figure 27: Key for User Levels

Key for User Levels:

Symbol	Number of RuleNodes Created
wood	>= 0
stone	>= 10
bronze	>= 20
iron	>= 30
silver	>= 40
gold	>= 50
platinum	>= 60

11.3 RuleNodes and the Rule Mesh

The following table provides definitions for key RuleNode and Rule Mesh Concepts used.

Table 14: RuleNode and Rule Mesh Concepts

Rule Tree or Decision Tree	A directed acyclic graph with rules at each decision node as described in Chapter 4 for traditional MCRDR systems. This structure was used in the FastFIX prototype.
Condition Mesh or Decision Mesh or Rule Mesh	The traditional MCRDR decision tree with the additional property that each child RuleNode may have one or more parent RuleNodes. The rule mesh need not be acyclic. This structure is proposed for future system embodiments as described in section 13.4 on page 245.
RuleNode or ConditionNode	One of the nodes in the rule tree or condition mesh. RuleNodes contain a Rule Statement, which is a boolean expression that may include pattern matching, comparative, or custom operators that can be evaluated by the system to determine the truth of that rule statement for a given case. Each RuleNode in the system is given a unique integer identity.
Live RuleNodes	A Live RuleNode for a given Case is one that is currently the last `TRUE` RuleNode on a given path through the knowledge base for that case. Its conclusions are part of the set of current conclusions derived from the knowledge base for the case. The system remembers its Live RuleNodes for "Tracked" cases as described in section 11.5 on page 199. Live RuleNodes may be correct or incorrect. Correcting incorrect live RuleNodes is the primary role of a (human or computer) expert who trains the knowledge base and hence builds up the knowledge in the rule tree.
Registered RuleNodes	A Registered RuleNode is one that has been confirmed by a User as being correct and `TRUE` for that Case. For each case, each RuleNode registration may be current or expired. This is explored in more detail in section 11.3.2 on page 185, Appendix O.2 on page 452, and Appendix O.3 on page 453.
RuleNode Status	RuleNodes may have several states, including pending (i.e. awaiting approval), approved, or disapproved. Note that if a RuleNode is disapproved, then the RuleNode is stopped by setting the rule condition to `FALSE`. This is explored in more detail in Appendix O.5 on page 456.

continued overleaf…

Stopping RuleNode	A stopping RuleNode has a single conclusion that is the keyword "stop". Stopping RuleNodes can be used to invalidate a parent RuleNode by causing cases that are live for the parent RuleNode to drop-through to the child RuleNode. Stopping RuleNodes specify the conditions under which a conclusion should not fire (Edwards, 1996, p 178). A special type of stopping RuleNode is an unconditional stopping RuleNode. This is a stopping RuleNode with a rule condition that returns `TRUE` for every case evaluated against the RuleNode and it has a "stop" conclusion. For most user types, the system will not display live Stopping RuleNodes for a case in the Case View, unless they are also registered for that case. If however the Stopping RuleNode is registered for a case then it will be displayed.
Stopped RuleNode	A RuleNode with its Boolean test equal to `FALSE` for every case evaluated against the RuleNode is a stopped RuleNode. It cannot be *live* for any cases.

11.3.1 Live and Dependent Cases and RuleNodes

In the FastFIX prototype, at every RuleNode in the system a Live Case List (LCL) is recorded containing all of the cases currently *live* for the RuleNode. Similarly, at every Case in the system a Live RuleNode List (LRL) is recorded, containing all of the RuleNodes currently *live* for the Case. As well, at every RuleNode in the system a Dependent Case List (DCL)[170] is recorded containing all of the cases currently *live* for the RuleNode, or any of its dependent (i.e. child, grandchild etc) RuleNodes.

Note that Beydoun, Kwok and Hoffman (2000, p4) have previously defined the DCL for the parent of RuleNode *r* as *context(r)*, the DCL for the RuleNode *r* as *domain(r)*, and the LCL for the RuleNode *r* as *scope(r)*.

The LCL, LRL and DCL mechanisms were implemented as simple lists in the FastFIX prototype. As mentioned in Appendix O.2 on page 452, for a production system with potentially large volumes of Cases and RuleNodes a different data structure, such as a multi-level hashing structure, would be warranted.

[170] Note that in an alternate 7Cs system embodiment, the DCL for each RuleNode could be constructed on the fly from the LCL at the RuleNode and the LCL of each of its dependent RuleNodes.

Note that in traditional MCRDR systems, the DCL was generated on the fly as needed. It consisted of the cornerstone case for the RuleNode and each of its dependent RuleNodes, and it was known as the cornerstone case list. Note also that in traditional MCRDR systems, only one cornerstone case was kept at each RuleNode.

In the 7Cs system, multiple tracked cases can be live and possibly even registered for a given RuleNode in FastFIX. Some or perhaps even all of these cases can be applied as cornerstone i.e. exemplar cases for this RuleNode. In the FastFIX prototype, the cornerstone case list was constructed from the LCL at the RuleNode.

11.3.2 Registered Cases and RuleNodes

In the FastFIX prototype, the final set of RuleNodes confirmed as being `TRUE` by a user are *registered* for the case under review. As an example, when Case 101 in Figure 1 on page 42 was evaluated against the rule tree in Figure 5 on page 52 the set of last `TRUE` RuleNodes for each path through the rule tree was RuleNodes 1, 5, and 7. So if a user had agreed with all the conclusions presented then the *registered* RuleNodes for Case 101 would have been {1, 5, 7}.

At every RuleNode in the system a Registered Case List (RCL) is recorded containing all of the cases currently registered for the RuleNode. Similarly, at every Case in the system a Registered RuleNode List (RRL) is recorded, containing all of the RuleNodes currently registered for the Case.

As described in Appendix O.2 on page 452, in future embodiments of the system, the name of the user who registered each RuleNode as being correct for the case could be recorded, together with the most recent date and time at which the RuleNode – Case association was created or modified.

As with the live Case-RuleNode associations, for a production system with potentially volumes of Cases and RuleNodes, shared hash tables rather than separate lists could offer faster processing and simpler management of transactions and rollbacks. This is explored in more detail in Appendix O.2 on page 452.

(Note that in FastFIX, RuleNodes with getAttribute() or setAttribute() conclusions should not be registered for cases, since once the new information is added to the case it will fetch a deeper and different set of conclusions. Figure 26 on page 182 has an example of a getAttribute() conclusion in RuleNode 7).

11.3.3 The "Rule Tree" View

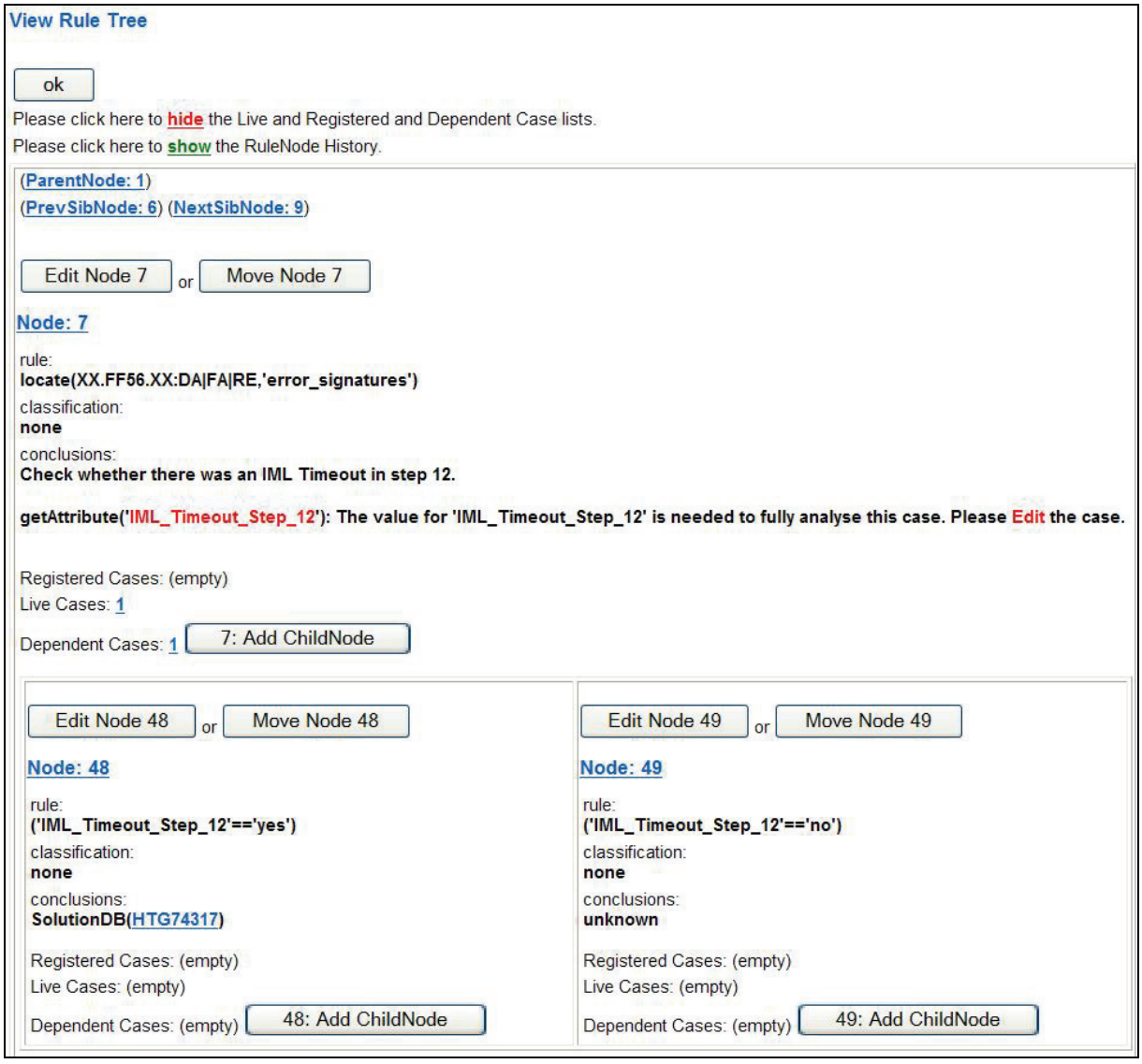
Figure 28 shows the Rule Tree View implemented in the FastFIX prototype. Items in blue and underlined are hyperlinks that can be clicked on to display an appropriate view of that element. RuleNode 7 is shown. In the prototype a decision tree is used so that child RuleNodes only ever have one parent RuleNode. Hence a HTML table structure can be used to display the rule tree. For future embodiments, a shared child RuleNode structure is proposed as discussed in section 13.4 on page 245.

As shown in Figure 28, the lists of live and registered cases for this node can be optionally displayed, as can the RuleNode history. Clicking on any of the case hyperlinks will bring up the corresponding case view.

In Figure 28 we can see that RuleNode 7 is a child of parent RuleNode 1, a sibling of RuleNodes 6 and 9, and a parent of child RuleNodes 48 and 49. Clicking on any of the RuleNode hyperlinks will bring up the rule tree view for that RuleNode, hence one can navigate around the rule tree using these hyperlinks. RuleNodes can be edited by clicking on the corresponding "Edit RuleNode" button in the rule tree view.

Note that for a RuleNode to be *live* for any given case, all of its ancestor RuleNodes must be `TRUE` for that case, and all of its descendent RuleNodes must be `FALSE` for that case.

Figure 28: Rule Tree View for RuleNode 7



As shown in Figure 28, RuleNodes are comprised of one or more rule conditions, and they are associated with one or more classifications and conclusions. This have been previously explained with reference to "classes" in Figure 10 on page 98. The "conditions", "classifications" and "conclusions" associated with each RuleNode and hence class are now described.

11.3.4 Conditions (i.e. Rule Conditions)

Rule Conditions are the primary means in ontology and descriptive logics by which exemplar cases i.e. instances are mapped to their abstractions i.e. their classes. They represent a

relationship between instances (the A-box[171]) and the classes (T-box) that represent them. In the FastFIX prototype, an LR parser[172] and rule compiler was implemented to evaluate the rule conditions created by users.

The operators shown in Table 15 were made available for users to create rule conditions in the FastFIX prototype. Note that in PEIRS, standard arithmetic operators were provided including a test for existence similar to that provided in the FastFIX system (Edwards, 1996, pp 82 – 83).

Table 15: Available rule operators

Operator	Description	Class
'==' or '='	EQUIVALENT TO	comparative
'!='	NOT EQUIVALENT TO	comparative
'<'	LESS THAN	comparative
'>'	GREATER THAN	comparative
'<='	LESS THAN OR EQUAL TO	comparative
'>='	GREATER THAN OR EQUAL TO	comparative
'&&'	BOOLEAN 'AND'	combinatorial
'||'	BOOLEAN 'OR'	combinatorial
'&'	BITWISE 'AND'	combinatorial
'|'	BITWISE 'OR'	combinatorial
!	NOT i.e. negation	unary
'?'	EXISTS	unary
'!?'	DOES NOT EXIST	unary
'grep('pattern','subject')'	GREP pattern modifiers include unix / perl / php grep modifiers. The 'subject' must provide the name of the attribute whose value is to be grepped.	functional
locate ('errorsig: parameters: directors', 'subject')	For example: locate(XX.FF56.XX:S12=01&S13=02:DA|FA,'error_signatures') will check that the error pattern "XX.FF56.XX" is present with sense byte 12 == 01 AND sense byte 13 == 02 on either a DA OR on an FA in the error signature provided.	functional

[171] Amongst many others, Giacomo & Lenzerini (1996) provide definitions of the A-box and T-box.

[172] An LR parser parses the input conditions in left-to-right order. Further information is available at http://en.wikipedia.org/wiki/LR_parser.

FastFIX supports the use of complex nested expressions using brackets. However the PEIRS trial found that complex expressions were only used in 6.1% of RuleNodes and were mostly used for representing time intervals and rates of change. In time they were abandoned in favour of more simply expressed heuristic rules that called upon predefined functions (Edwards, 1996, pp 115-117).

The *locate* operator shown in Table 15 provides an example of an HTG-specific predefined function that allows heuristic rules to be more simply expressed. The *locate* operator was created in light of the dial-home context described in Chapter 10 so that HTG's Hardware Support engineers could check for the presence of particular error codes at that RuleNode (XX.FF56.XX in the example), including particular values of sense bytes (S12 and S13 in the example), and on particular hardware directors (DAs or FAs in the example). For example, for RuleNode 7 as shown in Figure 28 on page 187, the rule is:

locate(XX.FF56.XX:DA|FA|RE,'error_signatures')

This checks whether an error matching the XX.FF56.XX error mask has occurred on a DA, FA, or an RE as recorded in the attribute entitled 'error_signatures'.

PEIRS included simple built-in functions such as current(), maximum() and minimum(); global functions such as netChange(), average(); and indexed functions that could be nested as parameters in other functions such as timeDifference() (Edwards, 1996, pp 79 - 81). PEIRS also supported the use of the clinical_notes field in a similar manner to the grep() function supplied in FastFIX. The use of the clinical_notes field increased over time and appeared in the rule conditions for more than 20% of RuleNodes in the final phase of PEIRS operation (Edwards, 1996, p 111).

In PEIRS, Edwards (1996, pp iv, 83, 145) showed that allowing the expert to define and re-use their own functions eased the rule-building task for pathologists. Re-useable functions were a feature in TCRDR and were seen to simplify and simultaneously increase the sophistication of feature definition for pathology data in PEIRS so that repetitious knowledge acquisition was reduced. This was important even in MCRDR systems, where repetitious re-definition of lengthy or complex rule expressions was still found to be a KA problem (p120). In future embodiments of the system, new functions could be configured or coded by the user.

As well, natural language processing could be used to translate user articulated rules into the system's rule syntax. For example, the natural language statement "that the product name

contains ‘product A’” may be translated into the rule expression: “grep(‘product A’,’product’)”. User configured templates, similar to those reportedly used in PKS’s LabWizard product (see section G.12, page 411), could be used to define these translations.

As well, in future system embodiments, functions used in the rule conditions could return multiple variables that could for example be used as input to other calling functions. This would allow a useful nesting of functions. The output of these functions could also be used and/or displayed in the conditions, classifications and/or conclusions of RuleNodes.

In future system embodiments, more complex rules could be analysed in a pre-processing step invoked for new or edited cases. This pre-processing step would set or clear boolean attributes for the case corresponding with the more complex rule statement being tested. This mechanism could be used to reduce the amount of text-based processing required by rules that may appear in more than one location in the FastFIX decision mesh. A pre-processing step similar to the one described here was used to extract important features from doctor’s comments and numeric data in cases analysed during the MCRDR trials involving the Garvan Thyroid database (Kang, 1995, p 29, 30, User Manual p3). As well, in PEIRS, data could be pre-processed as represented as qualitative descriptors such as HIGH, NORMAL, LOW, INCREASING, and DECREASING (Edwards, 1996, p82). Although over time in PEIRS it was found that users preferred to use numerical rule expressions rather than those referring to a pre-processed reference range (Edwards, 1996, PhD, p109).

Note that in conventional MCRDR, the conditions at RuleNodes were comprised only of conjunctions of rules (Kang, 1995 p18). This is one of the major reasons why repetitious knowledge acquisition was still a problem in conventional MCRDR systems. In the 7Cs and FastFIX model, the logical OR is permitted[173].

RuleNodes could also simply contain the logical `TRUE` or `FALSE` rule condition. As mentioned in Table 14, and Appendix O.5 on page 456, RuleNodes can be completely stopped by setting the rule condition to `FALSE`. This disables that RuleNode, and any of its

[173] Although (A || B) can be represented as the logical alternative !(!A && !B), the latter representation is not as readable, intelligible or useful as the former for domain-oriented human users. Permitting the logical OR makes sense when the heuristic identification of a given classification e.g. XYZ only requires that some of the features be available in the case being classified, for example when (XY OR YZ) results in the XYZ classification.

dependent RuleNodes, since no cases can be live for a RuleNode whose rule condition always returns `FALSE`, or for any of that RuleNode's dependent RuleNodes[174].

11.3.5 Classifications

In FastFIX, a classification is grouping, category or a set of things that share one or more common properties. It is a conceptual extension of a set of things and it is defined by a set of rules that all members of the classification obey. A classification is the result of a case evaluating to `TRUE` through a sequential set of RuleNodes, where each RuleNode has a Boolean test that may examine the attributes of the case. A classification may be labelled using text or hyperlinks, or it may remain unlabelled. Note that labelled classifications were known as an intermediate conclusions in N-RDR as described in section 4.4.5 on page 62.

In pathology, classifications may be diagnosis-oriented, for example: "kidney stones", "breast cancer", and "blood sugar too high". Examples of some labelled classifications in ICT support include the ones used in Figure 5 on page 52, namely: "product B", "windows platform", "old software".

Note that previous MCRDR implementations did not distinguish between classification labels and conclusions. Rather, previous MCRDR implementations required both classification labels and conclusions to be the conclusion at a RuleNode. In the 7Cs system, classification labels are explicitly allowed that are separate from other types of conclusions so that the classification labels can be referred to at other RuleNodes.

For the benefit of the HTG FastFIX software trial, the FastFIX prototype was customised to include only one classification label per RuleNode. However in general, classifications may have more than one label or name as in the sciences (geology, biology, physics, chemistry). For instance in botany, unique species of plants may have a scientific name, and a common name. Alternate embodiments of the system should allow room for multiple classification labels to be applied for each RuleNode. Where a RuleNode does have one or more identified classification labels, then any of the alternate classification labels can be referred to and reused elsewhere in the system.

---

[174] Note that this provides a different affect to having a RuleNode with a stopping conclusion. A stopping conclusion just prevents the RuleNode from being displayed under the prescribed rule conditions. The *stop* conclusion type is discussed in section 11.3.6 on page 193.

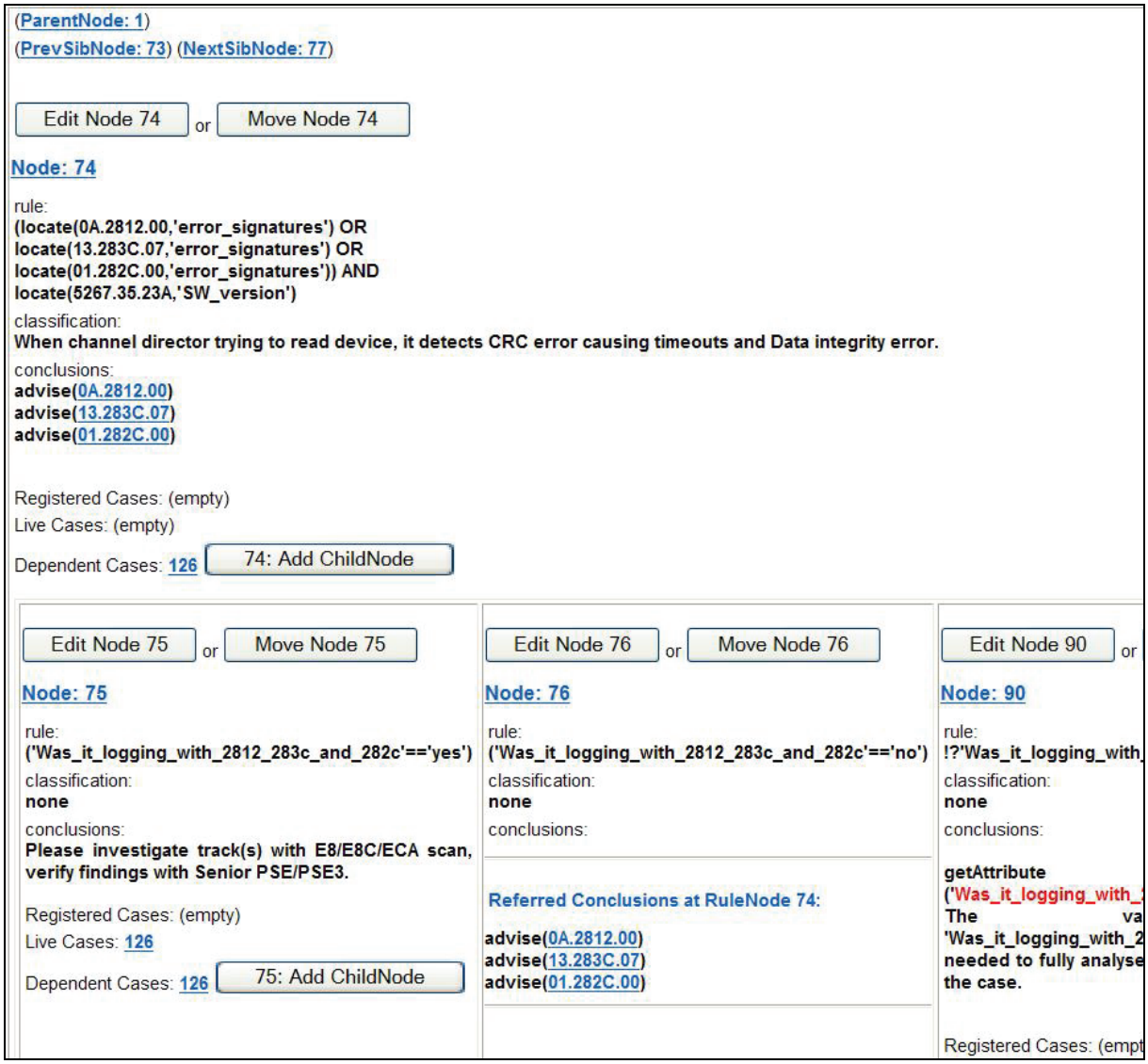
In many domains, an expert may not be able to think of an appropriate label or name for a classification. For instance in the health domain, perhaps the expert can observe a set of symptoms without knowing the exact disease. However, the expert knows what actions need to be taken as a result of arriving at a particular RuleNode, and the corresponding unlabelled classification. For example the actions may be that the patient should "drink more water", "cut out refined sugars", "eat more fibre" and so on. Providing a classification label at a given RuleNode is therefore optional.

Classifications from arbitrary RuleNodes in the 7Cs system can be pre-processed and presented in an integrated fashion as part of the conclusion view for the case by using a refer(RuleNodeID) function in the conclusion.

The following screenshot shows the facility to refer to arbitrary classifications in the FastFIX decision tree. In this figure, the conclusion at RuleNode 76 is refer(74). FastFIX interprets this and displays the conclusions at parent RuleNode 74 at child RuleNode 76.

Note that the advise() function provides hyperlinked text advice for the meaning of each of the >10,000 different error codes that may appear in the HTG dial-home logs.

Figure 29: Showing the conclusions from arbitrary RuleNodes in the decision tree



Additional examples of the benefit and power of classification reference and re-use are provided in section 13.6 on page 258 and section 13.7 on page 261.

11.3.6 Conclusions

In FastFIX, a conclusion represents one or more propositions, or final statements, including actions that one should take as a result of arriving at a given classification.

In pathology, conclusions may be prescription-oriented, for example: “eat 100g fibre bran for breakfast each morning”, “do 15 mins aerobic exercise per day”, and “drink 2L water per day”.

In I.T. support, conclusions may be action-oriented. The conclusions may include instructions for example to: click on a web link such as http://lookatme.pdf; run a particular software application with a particular set of parameters; read a document located in a particular place; ask the customer a particular question; or enter a value for a particular attribute. Conclusions can be a single word, a lengthy passage of text, or even a directive to use a particular Internet search engine with a suggested set of search criteria to further navigate the solution space.

Conclusions are statements that can include Internet URIs, plain text, or instructions to the system to interactively prompt the user for more A-V details and then recursively re-evaluate the case. In the context of HTG support centre problems, a typical approach would have conclusions that are sets of intranet URIs pointing to existing solutions in the SolutionDB solution repository or the document repository (Docco).

In the FastFIX prototype, a conclusion can be of any of the types that an attribute can be, including a hyperlink (see Table 17 on page 197 and Table 18 on page 198). Conclusions can also be one of the special types shown in Table 16 and described here:

- a stopping conclusion that prevents the RuleNode from being displayed under the prescribed rule conditions;
- a getAttribute('attributeName') conclusion that indicates that the user should be prompted to enter the value of a particular attribute, in this case the 'attributeName' attribute;
- a showfile() conclusion that provides a hyperlink to an uploaded file, or perhaps even displays the file's contents inline;
- an advise() conclusion that provides a hyperlink to the text advice for a particular HTG dial-home error code; and
- a refer() conclusion that refers to conclusions provided at another RuleNode or classification in the system.

Table 16: Available Conclusion Types

Conclusion Type	Example
Stopping Conclusion	stop
Get Attribute	getAttribute('habitat')
Show File	showfile(uploadedfileNum)
Show Error Code Advise	advise(XX.FF56.XX)
Refer to the Conclusions at another RuleNode	refer(RuleNodeID)

In future system embodiments a setAttribute('attributeName', 'attributeValue', confirm=true) conclusion could also be used to indicate that the 'attributeName' attribute should have its value set to 'attributeValue', that the changes should optionally be confirmed by a user, and that the case should then be re-evaluated against the rule tree. As mentioned previously (section 4.4.2 on page 61) in the ion chromatography solution reported by Kang (1995, pp132-133) an inference mechanism was developed that automatically filled in missing values in the SCRDR system to assist with the configuration of the ion chromatography equipment. The setAttribute() conclusion type would achieve the same result within a multiple classification paradigm. As well as that, it could be used to provide similar functionality to the hierachical nesting of concepts provided by NRDR (Beydoun and Hoffman, 1997). Note that N-RDR was previously described in section 4.4.5 (page 62). (Compton, Cao and Kerr, 2004, section 2.2 p4) also describe a process by which classifications add information to a case, with or without the input of a human user, and the case then gets re-evaluated against the KBS.

The lack of tools for abstracting specific features to more general features was seen as an important limitation in PEIRS (Edwards, 1996, pp 120, 134, 137) and is also discussed in (Richards, 1998a, pp 71-72). As discussed in section 4.4.5 on page 62, N-RDR offers an approach to overcome this limitation[175]. Alternatively, the setAttribute() conclusion could be

[175] As discussed later on (section 11.7.1 on page 210), N-RDR avoids the case drop-through issues suffered by MCRDR (section 6.5 on page 84) by insisting that any inconsistencies in the database resulting from changes to classifications are sorted out at the time of the database modification. 7Cs takes a much more flexible approach by allowing inconsistencies to be tracked via the live versus registered case-RuleNode associations so that inconsistencies in the database can be sorted out at a time that makes the most sense to the user.

used to provide a simple solution. The case would simply be evaluated against a relevant function / concept at the selected RuleNode. Then, if the function evaluates to `TRUE`, the conclusion at that RuleNode could set an attribute to store the derived value of that function / concept for the case in question. In order to consistently manage the evaluation of other RuleNodes for the case, the case would be recursively evaluated against the rule tree each time a setAttribute() function affected some change in the case.

Previous MCRDR implementations only allowed one conclusion per RuleNode As noted previously in section 11.3.5 (page 191), classifications are diagnosis-oriented whereas conclusions are prescription-oriented. Obviously in real-life, prescriptions (conclusions) can be shared across multiple different diagnoses (classifications) and in multiple different combinations. Therefore the 7Cs system allows multiple conclusions per RuleNode where those conclusions can be reused as one of several conclusions for multiple RuleNodes across the system. For some embodiments, the provision of multiple conclusions per RuleNode could help in the re-use of those conclusions by other users. As well, shared identical conclusions can be more easily kept up to date if they are only stored in one location, and a common alias or reference is used across the system for that conclusion.

In the FastFIX prototype the different conclusion types could be used repetitively in any combination within a given conclusion's text. For example observe multiple references to the SolutionDB() hyperlinked conclusion type in the hybrid text and hyperlink conclusion for RuleNode 15 in Appendix N (page 440).

In future embodiments of the system even richer conclusion types may be possible. These are described in section 13.7 (commencing on page 261).

Note that in the PEIRS system, users were encouraged to re-use existing conclusions. Alternate embodiments of the 7Cs system might encourage the re-use of conditions, classifications and conclusions by making them searchable, or by using a sentence complete facility as in many MS windows applications (Internet Explorer, MS excel), or by including them in a drop-through box, as appropriate.

11.4 Attributes

11.4.1 Attribute Concepts

In the 7Cs system, attributes may be binary, integers, floats, characters, strings, one of a set of strings/integers/floats, some of a set of strings/integers/floats, free text; or even combinations of the above types. In summary, attributes may be in the form of any of the standard data types one expects to see in a programming language, or as fields in a database; or some combination of those types. The following table shows some of the set and numeric types that are available when creating attributes in the FastFIX prototype.

Table 17: Set and numeric type attributes

Attribute Type	Description	Definition	Example
oneofset	one of a set	'class1','class2','class3'	An option select list is provided for available items in the set.
int selection	limited range integer	'min=0','max=100','step=10'	An option select list is provided for available integers in the set.
float selection	limited range floating point number	'min=0.0','max=100.0','step=10.0'	An option select list is provided for available floats in the set.
int freetext	raw integer		42
float freetext	raw floating point number		42.00

In addition, attributes may be hyperlinks, including hyperlinks to uploaded files. In alternate system embodiments, users could configure templates that translate a function used in a field of type 'hyperlink'. For example, the user could configure the syntax casetracker(101) used in an attribute (or a conclusion) for a case to translate to the hyperlink http://casetracker.com/mycasetracker.php?casenumber=101.

The following table shows some of the text and link types that are available when creating attributes (as well as conclusions) in FastFIX:

Table 18: Text and Link type attributes

Attribute Type	Description	Example
Text	Free Text	my example
URI	Internet web link	http://website.com.au
combo	A combination of free text interspersed with Internet URIs	checkout my website at: http://website.com.au and also checkout my friend's website at: http://myfriendswebsite.com.au
imageURI	web link to an image on the Internet	http://website.com.au/images/ *.jpg, *.jpeg, *.gif, or *.bmp
case()	link to a FastFIX case	case(caseNum)
CaseDB()	link to a CaseDB case	CaseDB(CaseDBNum)
SolutionDB()	link to a SolutionDB solution	SolutionDB(SolutionDBNum)
file()	link to an uploaded file	file(fileNum)
image()	link to an uploaded image file	image(fileNum)
refer()	link to an existing RuleNode	refer(76)

11.4.2 "View Attributes"

As shown in the following figure, each FastFIX attribute has a name; a type; a set of accepted-values (as required by one-of-a-set, some-of-a-set or ranged attribute types) and the attribute display units (for example kilograms, metres).

Figure 30: A View of Attributes in the FastFIX prototype

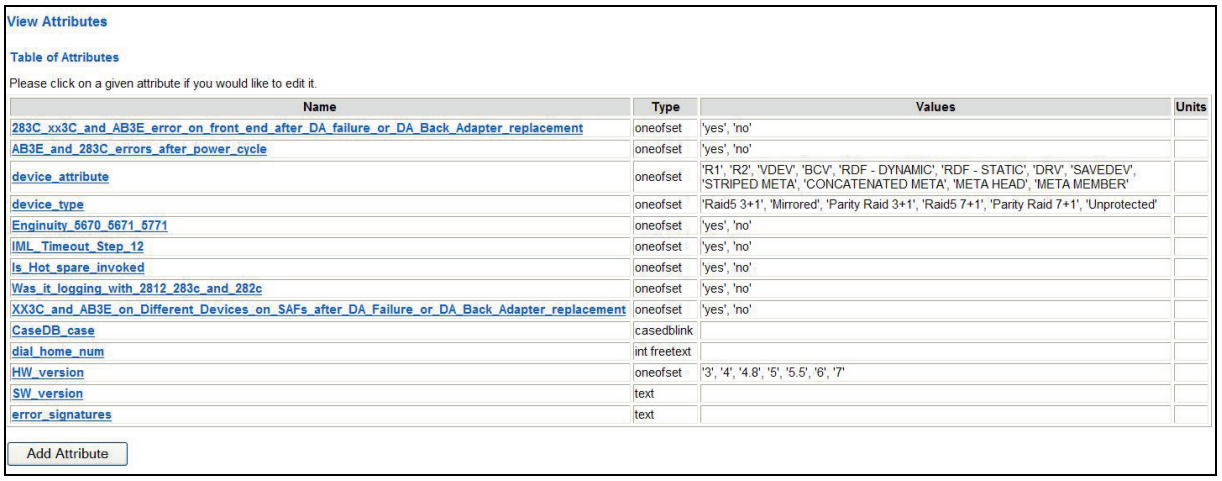

View Attributes

Table of Attributes

Please click on a given attribute if you would like to edit it.

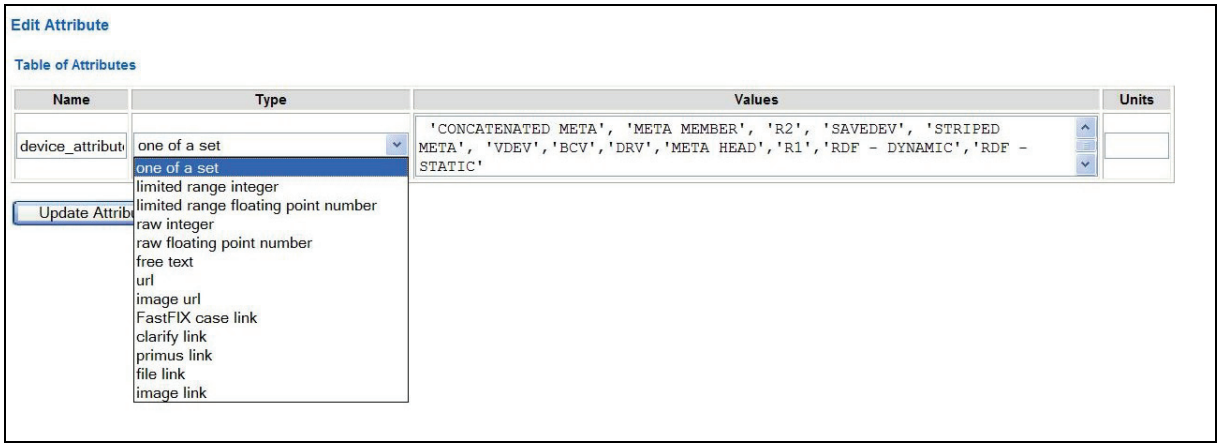
Name	Type	Values	Units
283C_xx3C_and_AB3E_error_on_front_end_after_DA_failure_or_DA_Back_Adapter_replacement	oneofset	'yes', 'no'	
AB3E_and_283C_errors_after_power_cycle	oneofset	'yes', 'no'	
device_attribute	oneofset	'R1', 'R2', 'VDEV', 'BCV', 'RDF - DYNAMIC', 'RDF - STATIC', 'DRV', 'SAVEDEV', 'STRIPED META', 'CONCATENATED META', 'META HEAD', 'META MEMBER'	
device_type	oneofset	'Raid5 3+1', 'Mirrored', 'Parity Raid 3+1', 'Raid5 7+1', 'Parity Raid 7+1', 'Unprotected'	
Enginuity_5670_5671_5771	oneofset	'yes', 'no'	
IML_Timeout_Step_12	oneofset	'yes', 'no'	
Is_Hot_spare_invoked	oneofset	'yes', 'no'	
Was_it_logging_with_2812_283c_and_282c	oneofset	'yes', 'no'	
XX3C_and_AB3E_on_Different_Devices_on_SAFs_after_DA_Failure_or_DA_Back_Adapter_replacement	oneofset	'yes', 'no'	
CaseDB_case	casedblink		
dial_home_num	int freetext		
HW_version	oneofset	'3', '4', '4.8', '5', '5.5', '6', '7'	
SW_version	text		
error_signatures	text		

Add Attribute

In the FastFIX prototype, all expert users were able to create and modify attributes.

The following figure shows the FastFIX view provided to users when they clicked on an attribute in order to edit it.

Figure 31: Edit Attribute View



Since URIs frequently expire, future systems would need to provide a facility for users to check the currency of URIs, and globally edit and update any expired web references.

As well, it should be possible to simply merge attributes that turn out to be synonyms of each other, and to update the global knowledge base accordingly. The first of the listed synonyms for a given attribute could be globally substituted throughout the system wherever any of its synonyms have been used. After that, hovering over that attribute with the mouse could inform the user of the complete synonym list for that attribute. Naturally an undo facility would be advantageous.

In future system embodiments a change history should be kept for attributes, identifying who made the changes and when, and perhaps with the ability to rollback changes as required. It would also be helpful to indicate to users the likely impact of their proposed attribute changes before the changes get committed to memory.

The super-user in the FastFIX prototype managed a manual display ordering of attributes. For future system embodiments, a hierarchy of attributes may be useful, as well as a configurable display ordering. An attribute hierarchy could be inferred from the decision mesh since higher-level attributes tend to be incorporated in the conditions for higher level RuleNodes in the decision mesh, and lower level attributes tend to be incorporated in the conditions for lower level RuleNodes in the decision mesh.

11.5 Cases

The following table provides definitions for key Case Concepts used.

Table 19: Case Concepts

Case	A Case is a set of one or more possibly temporal attribute-value (A-V) pairs that describes a user's observations of an object or a scenario. In defining the attributes of a case, users ask the question - what is our sense of the case? That is, how does the item look, sound, smell, taste, and feel? What are its properties? Cases and their attributes are observation oriented. For example, in geology a case may refer to a rock object with properties or attributes such as {weight, dimensions (height, width, depth), colour, composition, and temperature as measured every minute for the last 5 minutes}. The attribute will have corresponding values such as {950g, (100mm,50mm,25mm), grey, iron 98% nickel 1.5% and cadmium 0.5%, (100 deg C, 89 deg C, 80 deg C, 72 deg C, 64 deg C)}. Cases based on observations of an object may be observed in many domains where taxonomies naturally apply, for example: pathology, biology, botany, zoology, and chemistry. Figure 1 on page 42 showed some examples of cases in the I.T. support domain. These cases represent a trouble-shooting scenario being faced by a customer or employee of an I.T. organisation. Cases based on observations of a scenario might also be used in the legal or medical domains. Cases need not remain static and may represent observations of an object or a scenario over time. Cases may contain a history of case statements that have been added to the case over time by users. Cases are given a unique integer identifier in the FastFIX system.
Case Workflow Status	In future system embodiments, the status of cases may be recorded for example as: new, assigned, open, resolved, verified, closed, expired according to the desired workflow through the system. These state based case properties are in current use by multiple vendor case tracking solutions as previously described in section 2.4.1 on page 17. A new case is one that is new in the system and has not yet been assigned. An assigned case has been assigned to a user for the user to work the case and attempt to find a solution. An open case has been opened, i.e. accepted by a user. A resolved case has been solved by a user, i.e. a solution has been found for that case. A verified case has been verified as being correct by some user, possibly a user with extra "verify" privileges compared to a regular user. A closed case has been closed by a user. An expired case has been in the system so long that the system has decided to mark the case as expired.

continued overleaf...

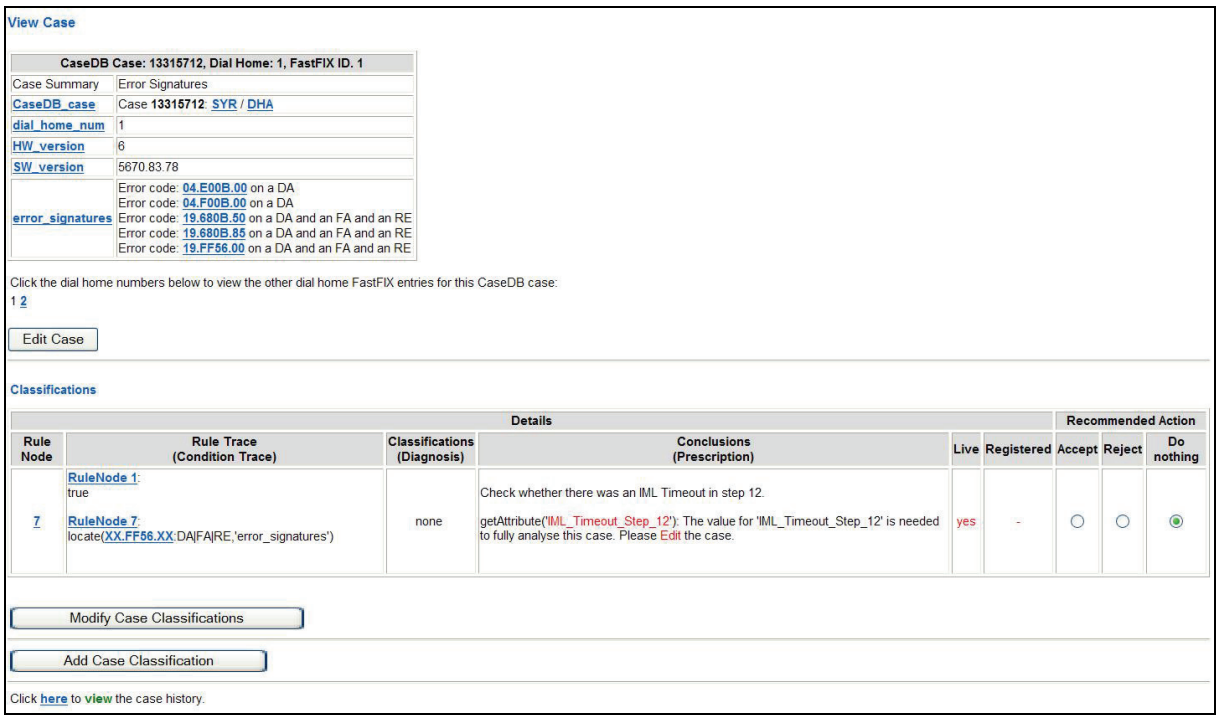
Live Case	A Live Case for a given RuleNode is one for which that RuleNode is currently the last `TRUE` RuleNode on a given path through the knowledge base for that case. Live Cases may be correctly or incorrectly classified by a given RuleNode. Correcting incorrect live Case-RuleNode associations is the primary role of a (human or computer) expert who trains the knowledge base and hence builds up the knowledge in the rule tree.
Registered Case	A Registered Case is one that has been confirmed by a User as being correct and `TRUE` for a given RuleNode. For each RuleNode, each case registration may be current or expired. This is explored in more detail in section 11.3.2 on page 185, Appendix O.2 on page 452, and Appendix O.3 on page 453.
Tracked Case	The notion of "tracking" cases is introduced by the 7Cs system. In this system, a "Tracked" Case is one whose Live and Registered RuleNodes are currently being recorded and remembered by the system. These case-RuleNode associations may be recorded in a Live RuleNode to Case association data structure, or a Registered RuleNode to Case association data structure, or in an LRL, LCL, RRL, or RCL as described in section 11.7 on page 210 and Appendix O.4 on page 454. In the FastFIX prototype, all cases seen by the system were *Tracked*.
Cornerstone Case	This is a case that has been stored by the system in association with one or more RuleNodes. It may be that when this case was executed against the knowledge base, one or more classifications given by the system were found to be in error, either by a human or software agent. As a result one or more new RuleNodes were added to the knowledge base to correct the classification and conclusions given. All of the RuleNodes that were added because of this case become associated with this case and likewise the case becomes known as the cornerstone case for these RuleNodes. Alternatively, the case may have been substituted as the cornerstone case for an existing RuleNode in accordance with section 11.7.3 (page 212). Cornerstone cases are retrieved and may be shown to an expert when the live RuleNodes for the case are deemed to be incorrect by some user. Cornerstone cases are used for system validation by providing the context behind a modification to the knowledge base and ensuring that new knowledge does not invalidate previous knowledge. Cornerstone cases also assist the user in the selection of features that will help form new rule conditions. Cornerstone cases provide the context and a counterexample for the new case that is motivating the knowledge in the system to be changed. Further information is provided in the Glossary (page 277).

11.5.1 "Case View"

In the following figure, a case view is shown for FastFIX Case ID 1. This FastFIX case was previously shown in Figure 23 on page 172 and refers to CaseDB Case ID 13315712 in Appendix B (commencing on p 387).

In the FastFIX prototype, the case view shows the attribute-value pairs comprised by the case, and the classifications and conclusions arrived at by the system for the case. For the benefit of the HTG implementation, the case view provides links to related FastFIX cases that record different dial-home events for the same CaseDB Case ID.

Figure 32: Case View for FastFIX Case ID 1: CaseDB Case ID 13315712 – Dial Home 1



In Figure 32 we see that the incoming case has been evaluated against the FastFIX decision mesh and there is only one live RuleNode for the case, namely RuleNode 7. The conclusion for this RuleNode is that the user needs to check whether there has been an IML timeout in step 12 of the equipment's start-up sequence.

The rule trace (condition trace) shows that the case arrived at RuleNode 7 by way of RuleNode 1. Note that for the case to be live at RuleNode 7 means that the case is `TRUE` for all of RuleNode 7's ancestors, and `FALSE` for all of RuleNode 7's descendents. As shown in Figure 28 on page 187, the descendents of RuleNode 7 are RuleNodes 48 and 49. In future

system embodiments it may be worthwhile displaying the negated descendent RuleNodes in the condition trace.

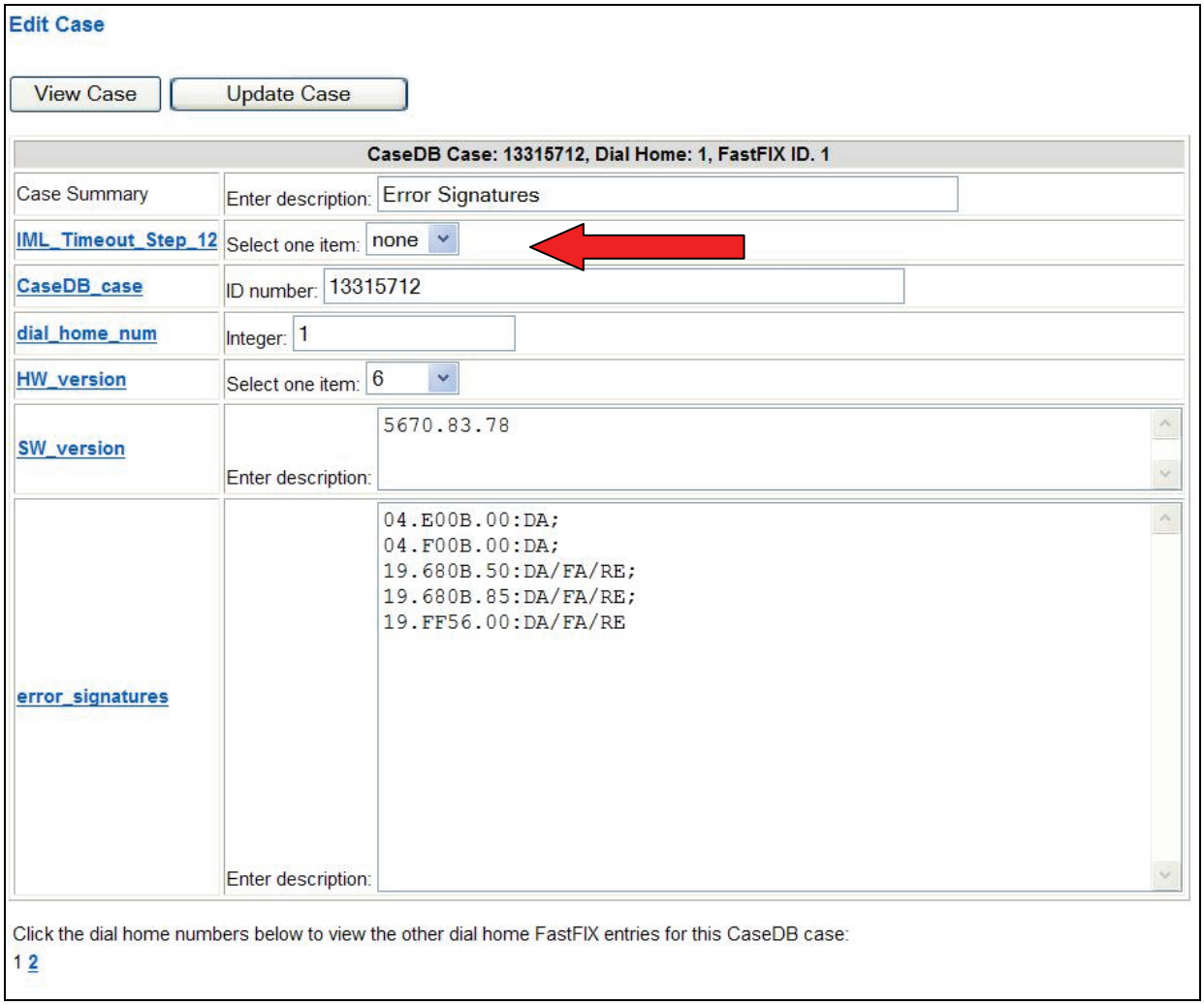
The system allows the live RuleNodes that have been evaluated for the case to be accepted or rejected or ignored on a *RuleNode by RuleNode basis* (see the "do nothing" column in the above figure). For example, the user can accept agreeable RuleNodes in the early stages of evaluating a case, and they can defer their decision on the contentious RuleNodes until a time when they can be more informed about the decision they need to make. The user can re-enter the case view at some later time to accept or reject the RuleNodes. When the user does decide to take action, the user can choose to "do nothing" for some RuleNodes, and separately "Accept" or "Reject" other RuleNodes.

In the example provided for FastFIX Case 1 in Figure 32, the user is advised to edit the case to answer the IML_Timeout_Step_12 question rather than accept or reject the RuleNode.

11.5.2 "Case Edit"

The next figure shows the result of selecting the "edit case" button in the case view. The user is presented with the case details and is able to edit them. In particular, they are asked to provide an answer as to whether there has been an IML timeout in step 12 via a corresponding parameter created by some user to reflect this question.

Figure 33: The "Edit Case" View



In future system embodiments, it might be more user friendly to record the question presented by the getAttribute('IML_Timeout_Step_12') conclusion in natural language and allow the user to answer the question in natural language inline within the case view. The attribute that stores the corresponding answer could be automatically generated and could remain hidden from users. A strategy would need to be developed to make sure that the same questions isn't asked multiple times (within the same or different RuleNode conclusions) within the same case view.

The Rule Tree View for RuleNode 7 presented in Figure 28 on page 187 provides a sneak preview of the possible results of further interaction by the user. If there has been an IML timeout in step 12, then the user is referred by RuleNode 48 via an Internet URI to the relevant solution in SolutionDB. Otherwise, the solution is unknown.

The above example demonstrates one of the key benefits of the 7Cs system – the ability for trouble-shooters to "work up a case" as described in the next section.

11.5.3 Working Up the Case

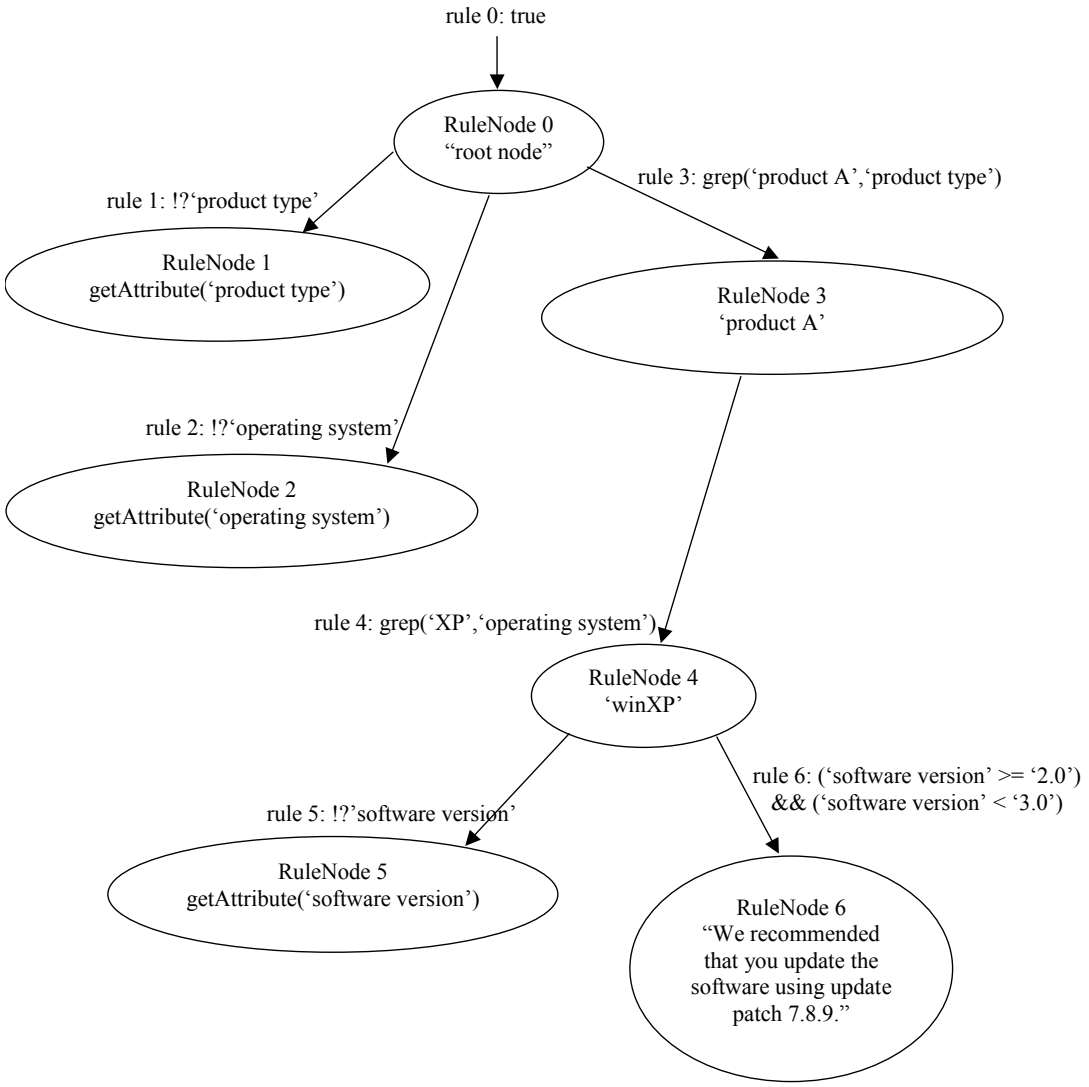
The idea of using MCRDR for propose and revise (Zdrahal and Motta 1995) construction tasks such as configuration has been previously proposed (Kang, 1995 pp ii, 131, 138), but as noted by Richards (1998a, pp 212 – 214) the work was postponed *"until the need became application driven"*. In the FastFIX prototype, two previously mentioned variations on the Single Classification RDR theme (section 4.4 on page 60) were adopted and integrated with the Multiple Classification RDR approach: Recursive RDR (R-RDR) (Mulholland et al 1993) that involved repeated inference cycles using the SCRDR structure; and Interactive RDR (I-RDR) which was a technique that allowed an SCRDR system to prompt users for more information when required.

Hence the 7Cs system provides a novel IR-MCRDR structure that allows users to "work-up" their cases. Importantly, rather than the user being overwhelmed with a large set of A-V pairs in the initial configuration of their problem case, they only need to enter A-V pairs that are relevant to the current case context. The effect is analogous to the Microsoft paper clip trouble-shooter where users are asked to answer a series of self-help questions that allows them to drill down through a decision tree to arrive at a useful solution. It also has parallels with user-interactions with an interactive voice response (IVR) system, for example the one provided by banks to access your account details, or by utility companies to pay your bills, where users are asked to make a series of keypad responses that allows them to arrive at some relevant conclusion or state in their query or session.

The difference in FastFIX is that the user can be simultaneously directed down multiple paths of the decision tree. As well, those paths are collaboratively and dynamically built by a group of domain experts. The net result is a data structure that allows cases to be configured and re-evaluated against a dynamically built decision tree.

A further example of working up a case is provided in the following figure.

Figure 34: Working up a case



In this example, the user brings an empty case to the system. The user is searching for a solution to the problem on hand. The customer is experiencing performance problems on a windows XP deployment of Product A with software version 2.5 and the user suspects that the customer needs to upgrade their software but they don't know the details.

Referring to the above figure, firstly, rule 1 tests that the 'product type' attribute has not been defined for the case. Note that in this syntax, !? tests that the RHS of the expression does not (!) exist (?). Obviously this is `TRUE` for an empty case so the user is prompted by the live RuleNode 1 to edit the case and enter the 'product type'. As well, rule 2 tests that the 'operating system' has not been defined for the case. This is also `TRUE` for the empty case so the user is prompted by the live RuleNode 2 to edit the case and enter the 'operating system'.

Assuming that the user edits the case and enters the product and operating system details as 'product A' and 'windows XP', now rule 3 and rule 4 both evaluate to `TRUE`. At this point rule 5 also evaluates to `TRUE` and the user is prompted to enter the 'software version'. Again, the user obliges and enters the software version as 2.5. The case then evaluates to `TRUE` for rule 6 and the user receives the recommendation: "We recommended that you update the software using update patch 7.8.9."

This example demonstrates several benefits of the 7Cs system. Firstly, since the rule tree is collaboratively built, a number of experts can document the relevant questions that a novice should ask themselves when attempting to define and classify the problem on hand and subsequently navigate the knowledge base. Secondly, for a novice user who may not have any clue where to start in attempting to find a solution to their problem, this mechanism provides a guided trouble-shooting methodology that leads the user through the rule tree to possible scenarios that the user may be experiencing, and possible solutions. Thirdly, the user only has to enter attributes relevant to their particular problem. This provides an effective and efficient way to manage potentially large lists of attributes for users, and it minimizes the time and effort that a user must expend in defining and classifying their problem, and finding a solution to it. Fourthly, since the user can be guided down several paths of the decision tree at once, they can potentially arrive at a relevant solution more quickly than if they could only traverse through a single decision path. Analogous examples could easily be constructed for other domains.

When the user attempts to first create a case, the near-empty rule base will not present any attributes for the user to enter. The user will first need to populate the rule tree with RuleNodes checking for a lack of existence of the relevant attributes, and concluding with getAttribute() statements defining those attributes that should be fetched at that level of the rule tree. Alternatively, if child RuleNodes test for some attribute which hasn't already been defined in a case the system could automatically create the requisite getAttribute() requests.

In summary, an interactive and recursive IR-MCRDR approach has been developed and tested that helps users to: collaboratively capture problem determination and classification determination knowledge; navigate the solution or classification space in the knowledge base; and quickly classify the case on hand and find matching conclusions for it.

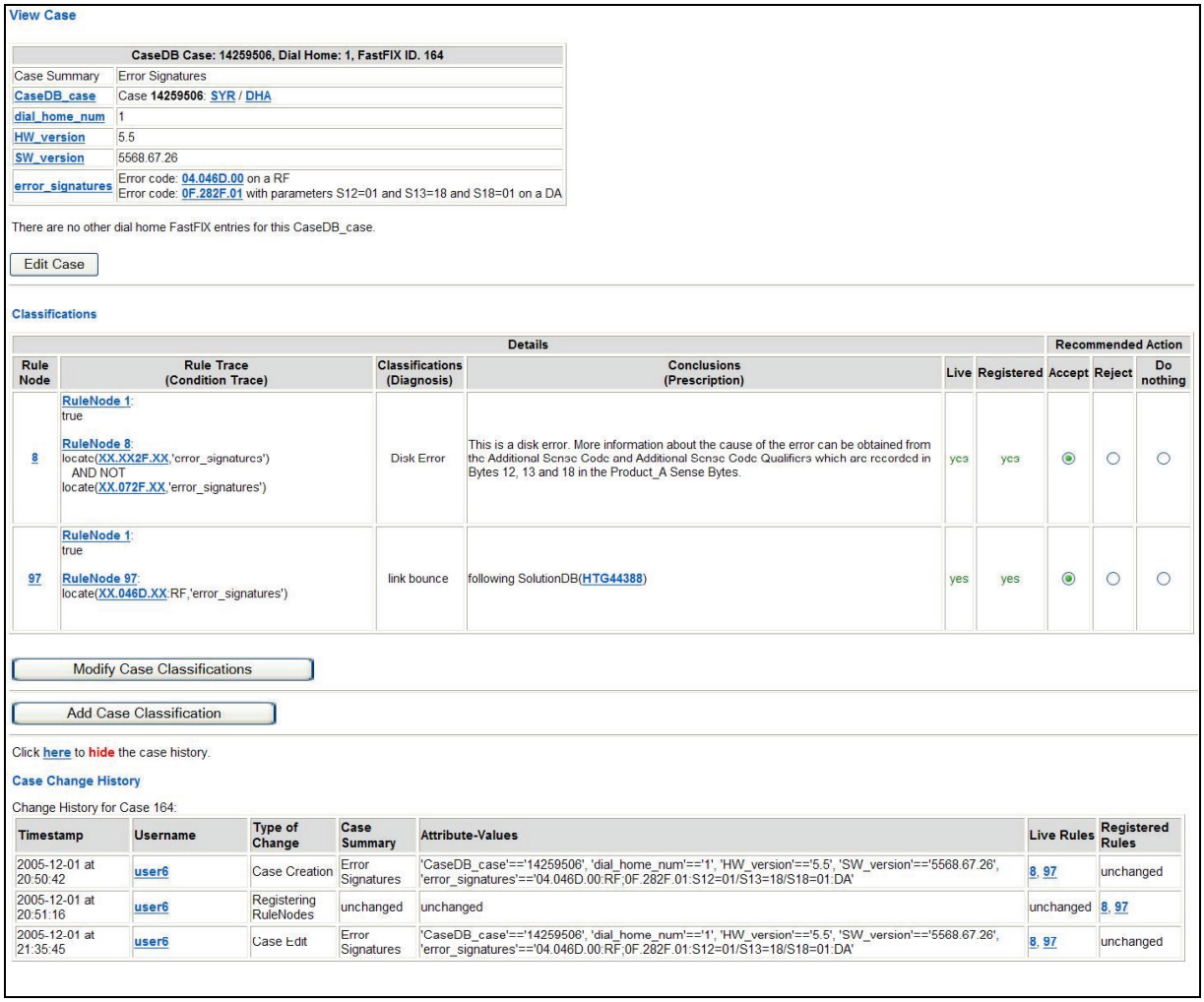
11.6 Change Histories

Change histories were not kept in the conventional MCRDR system trialled by Kang (1995, p141). Our interview with PKS indicated that the LabWizard product was recording some change histories, for example for conclusions (section G.12 page 411). However change histories were not kept for the case-RuleNode associations, since LabWizard only recorded historic snapshots of the cornerstone cases. At the time of this research, there was no facility in LabWizard for dynamic cases and their changing RuleNode associations to be tracked by the system.

In the FastFIX system, change histories are kept to identify the changes made, by whom and when. This is a key component of conflict resolution in the system. In the FastFIX prototype, the change history is kept for Cases, RuleNodes, and their associations; and in future system embodiments it should be kept for all attributes, conclusions, and rules (described in Appendix O.3 on page 453).

The following screenshot shows FastFIX Case 164 that is live and registered for RuleNodes 8 and 97 in the decision tree. The change history for the case shows that it was created, its live RuleNodes were registered, and then the case was edited.

Figure 35: Case View showing Change History for FastFIX Case ID 164[176]



View Case

CaseDB Case: 14259506, Dial Home: 1, FastFIX ID. 164	
Case Summary	Error Signatures
CaseDB_case	Case 14259506: SYR / DHA
dial_home_num	1
HW_version	5.5
SW_version	5568.67.26
error_signatures	Error code: 04.046D.00 on a RF Error code: 0F.282F.01 with parameters S12=01 and S13=18 and S18=01 on a DA

There are no other dial home FastFIX entries for this CaseDB_case.

Edit Case

Classifications

Details						Recommended Action		
Rule Node	Rule Trace (Condition Trace)	Classifications (Diagnosis)	Conclusions (Prescription)	Live	Registered	Accept	Reject	Do nothing
8	RuleNode 1: true RuleNode 8: locate(XX.XX2F.XX,'error_signatures') AND NOT locate(XX.072F.XX,'error_signatures')	Disk Error	This is a disk error. More information about the cause of the error can be obtained from the Additional Sense Code and Additional Sense Code Qualifiers which are recorded in Bytes 12, 13 and 18 in the Product_A Sense Bytes.	yes	yes	◉	○	○
97	RuleNode 1: true RuleNode 97: locate(XX.046D.XX:RF,'error_signatures')	link bounce	following SolutionDB(HTG44388)	yes	yes	◉	○	○

Modify Case Classifications

Add Case Classification

Click here to hide the case history.

Case Change History

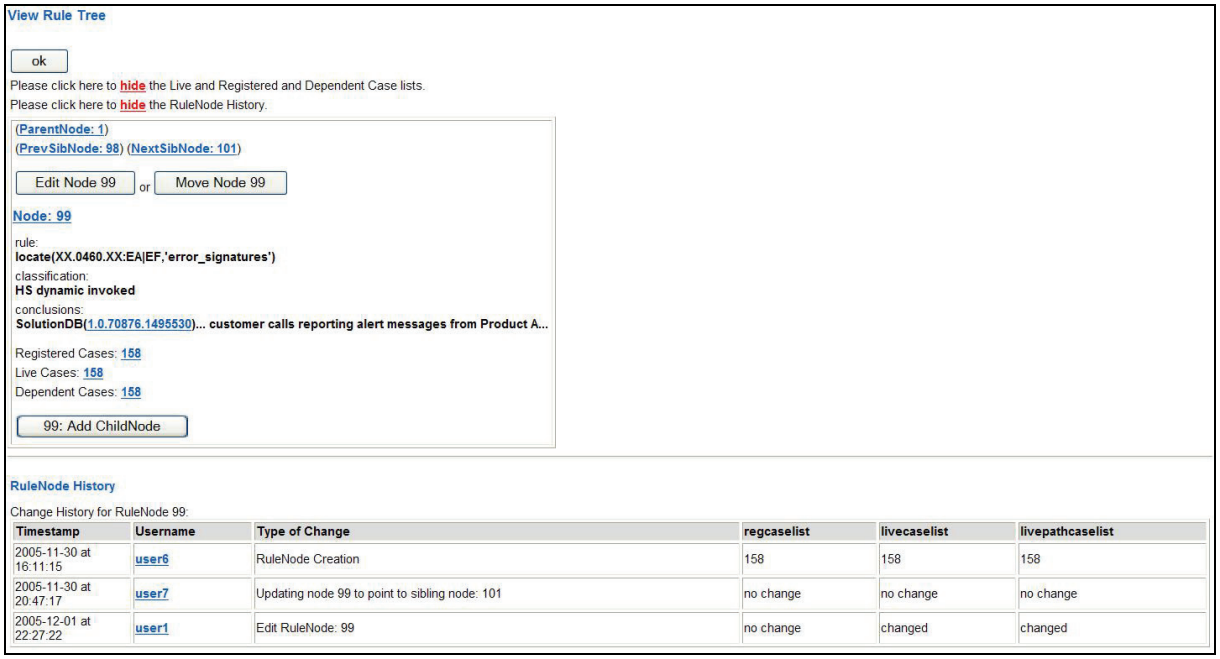
Change History for Case 164:

Timestamp	Username	Type of Change	Case Summary	Attribute-Values	Live Rules	Registered Rules
2005-12-01 at 20:50:42	user6	Case Creation	Error Signatures	'CaseDB_case'=='14259506', 'dial_home_num'=='1', 'HW_version'=='5.5', 'SW_version'=='5568.67.26', 'error_signatures'=='04.046D.00:RF;0F.282F.01:S12=01/S13=18/S18=01:DA'	8, 97	unchanged
2005-12-01 at 20:51:16	user6	Registering RuleNodes	unchanged	unchanged	unchanged	8, 97
2005-12-01 at 21:35:45	user6	Case Edit	Error Signatures	'CaseDB_case'=='14259506', 'dial_home_num'=='1', 'HW_version'=='5.5', 'SW_version'=='5568.67.26', 'error_signatures'=='04.046D.00:RF;0F.282F.01:S12=01/S13=18/S18=01:DA'	8, 97	unchanged

The following screenshot shows RuleNode 99 that is currently registered and live for FastFIX Case 158. The RuleNode history includes the history of which cases were live (see the livecaselist), registered (see the regcaselist) and dependent (see the livepathcaselist) on this RuleNode when the given event occurred, as well as who made the change and when it occurred. Activity from each of three different users resulted in the changes shown.

[176] The user name is aliased in the figure to protect the commercial confidentiality of HTG.

Figure 36: Rule Tree View for RuleNode 99



In future system embodiments zero-impact edits should be excluded from the change history for the sake of clarity and efficiency.

11.7 Case-RuleNode Associations

11.7.1 Relaxing the Case differentiation test

In the FastFIX prototype, the requirement to explicitly differentiate the current case from the set of cornerstone cases for a parent RuleNode is relaxed as it was done in PEIRS (Edwards, 1996, p184). This is because the case may have been inappropriately classified at the parent RuleNode. For example the case may now warrant a more specific classification than that provided by the parent node.

Relaxing the case differentiation test raises issues of knowledge consistency and version control since "case drop-through" may now occur. This is where a case in the DCL[177] at the parent RuleNode drops-down into the new child RuleNode and becomes live for that child RuleNode. The example provided in Figure 122 on page 468 illustrated the "case drop-through" situation in a traditional MCRDR KBS. Conventional MCRDR implementations did not handle this scenario well. For example, in PEIRS, the parent RuleNode would be left

[177] If required, please refer to the glossary on page 277.

without a representative cornerstone case since that case would drop-through to become the cornerstone case of the child RuleNode, and the case that caused the child RuleNode to be created would not be used as a cornerstone case for the new child RuleNode (Edwards, 1996, p184).

Chklovski identifies that a lot of burden on the enterer is generated when knowledge must be fully disambiguated at entry time (2001, p16). In N-RDR, abstracted RDRs can 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 cases (Beydoun and Hoffman, 1997) and (Beydoun et. al, 2005, p51)[178]. The potential complexity of this knowledge engineering task was previously highlighted in section 8.2.5 on page 137.

According to Suryanto (personal communication, June 2004) the FastFIX approach lies between NRDR and MCRDR:

- NRDR requires secondary refinement for affected cases,
- In FastFIX 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, and knowledge inconsistencies can occur.

In the support centre domain, there are very good reasons to delay the secondary refinement of affected cases: many of the users are novices, and much of the knowledge is dynamic, so it can take some time before the user knows what knowledge to enter, and how best to enter it.

11.7.2 Knowledge Evolution and Case Drop-through

In PEIRS, typographical or conceptual errors in RuleNode expressions could only be corrected with the use of "fall-through" rules (Edwards, 1996, p119). It was perceived that even subtle changes in RuleNode expressions would "corrupt the knowledge base".

The 7Cs system solves the problem of case drop-through that not only occurs when the knowledge base evolves, but also when a case is changed, when a RuleNode is changed, when an intermediate conclusion is changed, or when an attribute (or its ontological alias) is

[178] N-RDR was introduced in section 4.4.5, page 62.

changed. The separation of live and registered case-RuleNode associations is a key part of being able to resolve classification conflicts between multiple experts, and even between what a single expert thinks today, as compared with tomorrow (Gaines, 1993)[179].

In the FastFIX prototype, all cases presented to the system are *tracked* as defined in Table 19 on page 200. Appendix O.4 on page 454 describes the possibility of case un-tracking and re-tracking cases in future system embodiments.

The tracking of cases by FastFIX means that case drop-through scenarios can be identified and users can be made aware that they have occurred. The strategy assists with more rapid knowledge acquisition since it can highlight inconsistencies between expert opinions. It lets users capitalize on the knowledge acquisition opportunity presented by the case drop-through scenario, and this in turn can result in quicker coverage of the domain and greater learning opportunities for users. As noted by Easterbrook (1991, p26):

"*... the final resolution is not necessarily the most important product of the negotiation process – the extra information elicited during the process, and the participants new understanding of one another's viewpoints may be far more valuable*".

Note that cases don't only *drop-through* from parent RuleNodes. They can also drop-across from existing sibling nodes when the rule condition at the new RuleNode is permissive enough to also apply to cases associated with the existing sibling nodes; or they can *recoil* to a parent RuleNode for example when the editing of a child RuleNode is restrictive enough that it now excludes the case under review.

11.7.3 Cornerstone Replacement

When a RuleNode is augmented with a new child RuleNode, the FastFIX prototype will evaluate every case in the DCL at the parent node to see if it drops through to the new child RuleNode. If a parent cornerstone case drops-down to a new child RuleNode, it may be used as the cornerstone case for that RuleNode, and eliminated as a parent RuleNode cornerstone case. This is not an act of cornerstone compression (as in PKS US Patent: US6553361), but rather it is an act of achieving accuracy in the knowledge base. If a parent cornerstone case

[179] Dazeley and Kang (2004, p6) also refer to the *concept drift* problem.

drops through to the new child RuleNode, it is because in terms of knowledge acquisition, it no longer represents a valid cornerstone case for the parent RuleNode[180].

11.7.4 Knowledge Evolution Notifications

Differences in the LCL and RCL for each RuleNode, and in the RRL and LRL for each case were made available to viewers in the FastFIX Prototype via the "Check RuleNodes", "Check All Cases", and "Check My Cases" screens as shown in sections M.11, M.12, and M.13 (commencing on page 434).

In future system embodiments, users can be notified whenever knowledge evolution of interest to them occurs. The notification may be for example in the form of an email, an instant message, an SMS, or something similar.

For example, users could be notified whenever the RRL or LRL for a case that they are interested in changes. As well, the user could be notified whenever the LCL or RCL for a RuleNode that they are interested in changes.

11.7.5 Maintaining Knowledge Base Integrity

At some later stage the expert could view each affected case or RuleNode and train the system with the necessary knowledge to ensure that all the live case-RuleNode associations are registered and hence that the consistency of the knowledge across all the "Tracked" cases is maintained.

Importantly, users don't need to resolve differences between the RCL and LCL at a RuleNode, or between the RRL and LRL at a Case unless they specifically want to. Cases are allowed to hang around in the system in a state where their LRL is different to their RRL. In fact, users may decide to un-track such cases as described in Appendix O.4 on page 454, particularly if they have already been resolved and verified.

---

[180] In future system embodiments, the policy for setting the cornerstone case for new RuleNodes may be configured to be a FIFO (first in first out) or LIFO (last in first out) policy ordered for example by the age of "Tracked" cases in the system (or possibly just for "Tracked" cases that are resolved and/or verified) where the RuleNode in question is both registered and live for the replacement cornerstone case. If there are no such cases, then this policy can be relaxed in stages, for example to first look for a replacement cornerstone case from all verified cases that are live and registered, then all resolved cases that are live and registered, then any case that is live and registered, then perhaps just any case that is live.

11.7.6 Knowledge Acquisition Scenarios

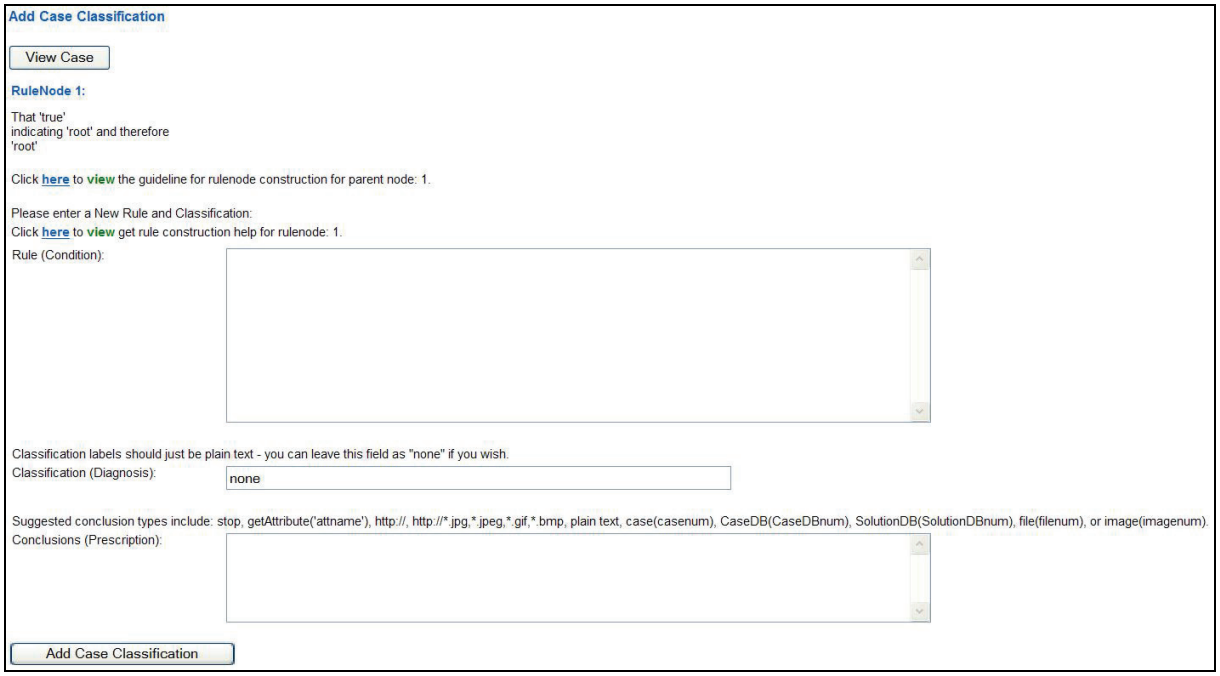
This maintenance of both live and registered case-RuleNode associations gives rise to the knowledge acquisition scenarios shown in Table 20. For example, the system evaluates an old case and shows the user that there is a registered RuleNode that is no longer live – perhaps the case has been edited or the case now falls down to a new RuleNode – 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*. In the table, 0 represents `FALSE`, and 1 represents `TRUE`.

Table 20: A Truth-Table of Knowledge Acquisition Scenarios

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

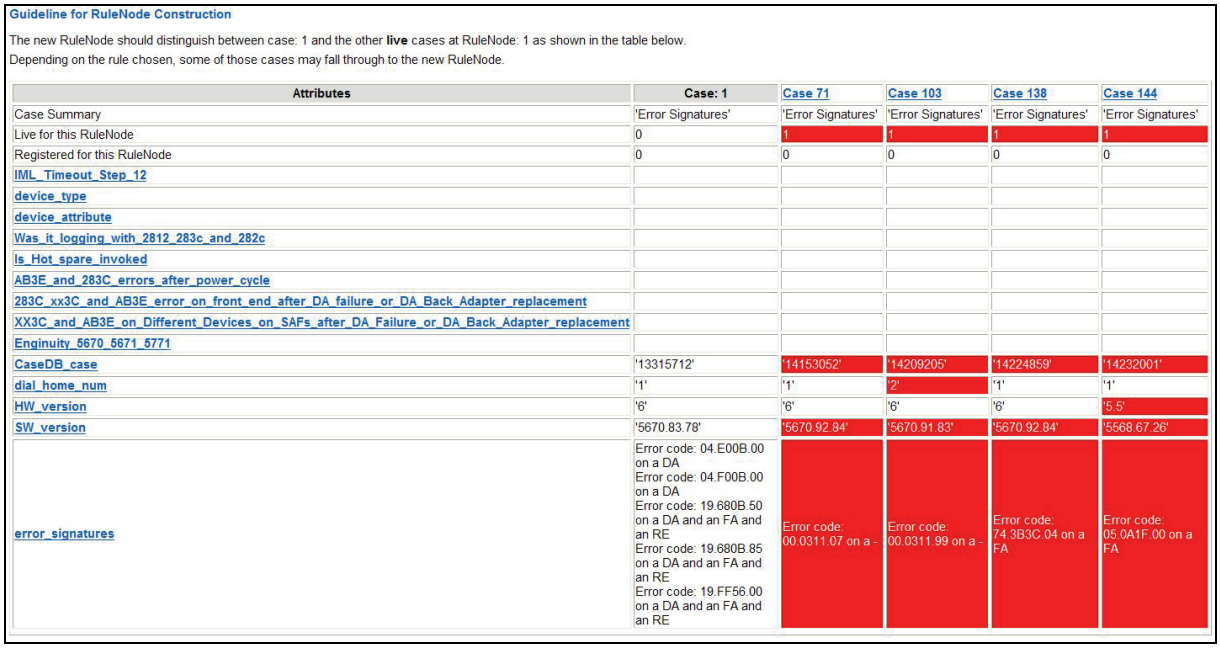
The following screenshot shows the interface in the FastFIX prototype for creating a new RuleNode. As can be seen in this figure, the main portions of the RuleNode that need to be created include its rule condition, optionally one or more classification labels, and the conclusions. In this figure the user has decided to add a new independent RuleNode at the top of the rule tree, hence its parent node is RuleNode 1.

Figure 37: Creating a New RuleNode with parent RuleNode 1



When the user chooses to view the guidelines for RuleNode construction for the parent node, they are provided with a table of cases in the LCL for RuleNode 1 as shown in the following figure. Differences between the attributes for these cases are highlighted in red so that users can easily construct a rule that differentiates from either all of these cases, or a subset of them. Undifferentiated cases will drop through to the new RuleNode.

Figure 38: Creating a rule that optionally differentiates from the LCL at RuleNode 1[181]



Guideline for RuleNode Construction

The new RuleNode should distinguish between case: 1 and the other **live** cases at RuleNode: 1 as shown in the table below.
Depending on the rule chosen, some of those cases may fall through to the new RuleNode.

Attributes	Case: 1	Case 71	Case 103	Case 138	Case 144
Case Summary	'Error Signatures'	'Error Signatures'	'Error Signatures'	'Error Signatures'	'Error Signatures'
Live for this RuleNode	0	1	1	1	1
Registered for this RuleNode	0	0	0	0	0
IML_Timeout_Step_12					
device_type					
device_attribute					
Was_it_logging_with_2812_283c_and_282c					
Is_Hot_spare_invoked					
AB3E_and_283C_errors_after_power_cycle					
283C_xx3C_and_AB3E_error_on_front_end_after_DA_failure_or_DA_Back_Adapter_replacement					
XX3C_and_AB3E_on_Different_Devices_on_SAFs_after_DA_Failure_or_DA_Back_Adapter_replacement					
Enginuity_5670_5671_5771					
CaseDB_case	'13315712'	'14153052'	'14209205'	'14224859'	'14232001'
dial_home_num	'1'	'1'	'2'	'1'	'1'
HW_version	'6'	'6'	'6'	'6'	'5.5'
SW_version	'5670.83.78'	'5670.92.84'	'5670.91.83'	'5670.92.84'	'5568.67.26'
error_signatures	Error code: 04.E00B.00 on a DA Error code: 04.F00B.00 on a DA Error code: 19.680B.50 on a DA and an FA and an RE Error code: 19.680B.85 on a DA and an FA and an RE Error code: 19.FF56.00 on a DA and an FA and an RE	Error code: 00.0311.07 on a -	Error code: 00.0311.99 on a -	Error code: 74.3B3C.04 on a FA	Error code: 05.0A1F.00 on a FA

Note that during the FastFIX trial, as also experienced and commented on by PKS (see the comment that "*Difference lists are rarely used since most users know what conditions they want to apply without needing to see a difference list*" in section G.12, page 411), this differentiation matrix was not used much in practice by experts populating the knowledge base. Experts generally knew off the top of their heads or on the basis of the case under review, the feature that would be relevant in characterising the new RuleNode that they wanted to add. As noted in the evaluation of PEIRS, derived values for functions did not appear in the difference lists, so valid rule conditions typically had to be created rather than simply selected (Edwards, 1996, p144).

In the LabWizard TCRDR-like MCRDR implementation of PEIRS, clicking on an attribute in the current case popped up a difference list for that attribute that included the evaluation of relevant functions that would differentiate the case against the cornerstone cases for the parent RuleNode (Edwards, 1996, p183). As well, Edwards suggested the use of a cornerstone profile, rather than a single cornerstone case (Edwards, 1996, p219) where the profile represents the full range of acceptable cases at that RuleNode. Edwards proposed that this

[181] In future system embodiments, attributes not referred to by either the case in question, or by cases in the LCL of the parent RuleNode, should be eliminated from this display.

information also be maintained in the context profile for the Feature Exception Prudence (FEP) feature described in section 13.8.1 on page 264.

Note also that attributes representing the pre-processed results of more computationally expensive functions like the locate() or grep() function would allow users to more simply distinguish these cases from each-other. Future system embodiments might allow multiple values from such function calls to be returned to several different attributes in the pre-processed case.

Note that in the FastFIX system, the user could transition from the view and hence state indicated by Figure 38 on page 216 to either add or remove attributes or their values, or to return to the case under review and edit it.

In conventional MCRDR, cases in the DCL of the parent RuleNode were represented in the difference list, rather than cases in the LCL that is a subset of the DCL at the parent node (Kang, 1995, p65). If a human expert were to have used Kang's proposed MCRDR system, given that there was no facility for cornerstone cases to be replaced by more representative cases as the knowledge evolved, cornerstone cases for stopping child RuleNodes (that accounted for 40-60% of all the RuleNodes acquired across the 3 trial domains – see pp 117 – 119) would have been included in the difference lists at the parent RuleNodes, even though they clearly do not represent good exemplar cases for the parent RuleNode.

In contrast, in FastFIX, only *live* cases at the parent RuleNode i.e. cases in the LCL are deemed to be representative cases for the parent RuleNode. Using the LCL instead of the DCL at the parent RuleNode helps to minimises the size of the difference lists and hence alleviates some of the concerns previously highlighted by Kang (1995, pp 108, 129) regarding the size of the KA task.

Note that cases arriving at an existing child RuleNode of the current parent RuleNode may validly fetch the classification offered by the proposed new child RuleNode. Hence it is suggested that in future system embodiments, that the user be forewarned if any cases in the DCL of the parent RuleNode, but not in its LCL are going to fall across to the proposed new child RuleNode[182]. As well, the user should be separately forewarned if any cases in the

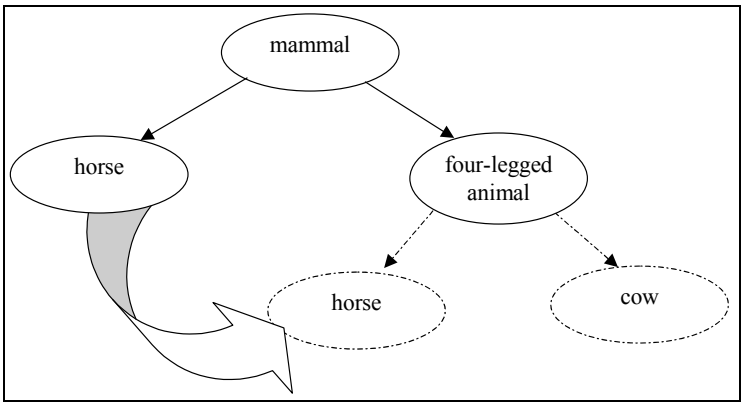
[182] If the new RuleNode is not mutually exclusive from its sibling RuleNodes, for example it is more general, then cases in existing sibling RuleNodes recorded in the DCL of the parent RuleNode, but not in the parent

difference list from the LCL of the parent RuleNode are going to fall into the proposed new child RuleNode under construction.

11.7.7 Moving RuleNodes

The facility to move RuleNodes was provided during the latter stages of the FastFIX software trial and an example is provided in Figure 39.

Figure 39: Moving RuleNodes



The following screenshot shows the RuleNode view with the Move RuleNode facility implemented. Note that when a RuleNode is moved, all of its descendent RuleNodes are moved along with it.

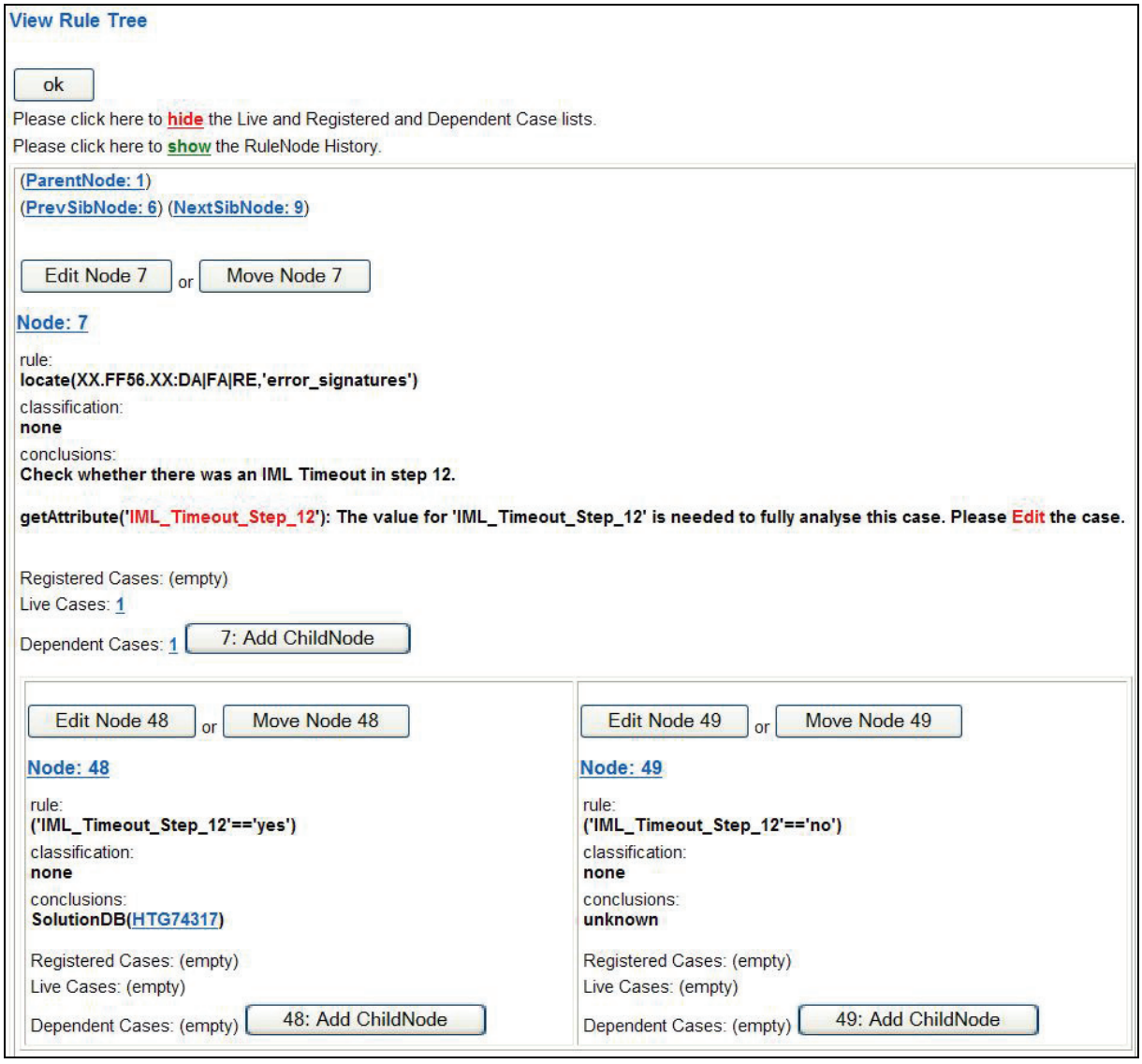
RuleNodes LCL and therefore not shown in the differentiation matrix, can drop down into the new RuleNode from the parent RuleNode's DCL. Such cases will fetch an additional classification and conclusion set.

For example, if the parent RuleNode represents mammals, and it already has a child RuleNode representing horses, then if a new child RuleNode is created for four legged animals with a cow case in mind, horses already classified by the system will not appear in the case differentiation list for the mammal RuleNode, but they will drop into the new four-legged animal RuleNode from the mammal RuleNode's DLC, so that they now fetch two separate classifications: horse and four-legged animal.

An astute user might decide to move the horse sub-classification underneath the four-legged animal sub-classification and also create a cow sub-classification, as discussed in the next section.

In a different example, the new RuleNode may not necessarily cover its sibling RuleNodes, but rather it may have some amount of overlap together with some amount of exclusion. In that case, some cases in the intersection of both sibling classifications will correctly fetch both sets of conclusions. For example, the animal and plant kingdoms may be mutually exclusive siblings in a KBS concerning life on earth, but some grey and overlapping regions may appear in the taxonomies of bacteria, yeasts, lichen, moulds, algae, and fungi.

Figure 40: RuleNode View with Move RuleNode facility



The next screenshot shows the panel used to determine where to move the RuleNode.

Figure 41:Details for Moving the RuleNode

Along with the facility to edit RuleNodes, the facility to move RuleNodes appears to be important in solving problems of over- and under-generalisation and hence problems of false

positives and false negatives in the KBS. When a RuleNode is moved, the old DCL for that RuleNode and the DCL for the new parent RuleNode can be used to update all of the affected LCLs, LRLs, and DCLs.

11.7.8 Case Edits

The FastFIX prototype allows users to directly edit cases, including cornerstone cases. This is a novel feature of the system. Except for cornerstone cases, traditional MCRDR systems do not keep a record of cases seen by the system. As well, traditional MCRDR systems do not allow cornerstone cases to be changed. If a case is edited in a traditional MCRDR systems (PEIRS and LabWizard), it is presented as a brand new case to the system.

In contrast, when a "Tracked" case is edited and changed in the FastFIX prototype it gets re-evaluated against the rule tree and a different set of Live RuleNodes may be derived for the case. Hence the LRL for the case may change, the LCL for each previously associated live RuleNode may change, and the LCL for any new newly associated live RuleNodes will change.

11.7.9 RuleNode Edits

As mentioned previously, in the FastFIX prototype, users are allowed to directly edit RuleNodes, including their rules, classifications and conclusions. This is another novel feature of the system. When a RuleNode is changed, the old DCL for that RuleNode and the DCL of the parent RuleNode can be used to update all of the affected LCLs, LRLs, and DCLs.

The ability to edit RuleNodes and Cases is a vital part of maturing the knowledge base. In this research it was found that human experts can and do notice errors that may not have been considered by previous machine learning approaches to simulating expertise (Compton et al. 1995) (Compton et. al, 2000), for example structural errors leading to over and under generalisations for ***as yet unseen cases***; wrong classification labels and wrong conclusions; the need for new or modified attributes and value options in the domain model; when a cornerstone case is no longer representative of the classification that it is attached to; special situations when RuleNodes should be added, deleted, edited, or moved.

Providing the *RuleNode and Case edit*, and *RuleNode move* top-down features can ease the knowledge acquisition process for human users. This was also the experience at PKS (see the

comment about the need for a balance between structured and unstructured KA, and the comment that "*Incorrect rules are never changed since rules are cheap. However PKS is looking at relaxing those conditions"*, in section G.12, page 411).

11.7.10 Optimised Data structures

Given the large number of cases that need to be handled, the underlying data structure is critical. In the FastFIX prototype, the following optimisations were made:

- Whenever a new RuleNode is added, or a Case, or RuleNode is edited, the system updates its internal LCL and LRL references to maintain database integrity. The enormous benefit of this is that when a tracked case is viewed, the case does not have to be re-evaluated against the rule tree because all of its live RuleNode to Case associations are already known and up to date. Similarly, when any RuleNode in the system is viewed, its live and registered cases are already known, and do not require evaluation. The system is a finite state machine whose live case-RuleNode associations for tracked cases are always known, and always current.
- When a new RuleNode is added, the system need only evaluate cases in the DCL of the immediate parent node. This saves significantly in processing overhead by reducing the number of cases examined.
- Each version of each tracked case is only evaluated against the decision tree or decision mesh once, and the results are recorded in the LRL for the case. Likewise, each version of each RuleNode is only evaluated on a once-only as-needs basis for the affected cases, and the results are recorded in the LCL for the RuleNode.
- The RRL is kept, even for untracked cases, so that when even when untracked cases are viewed in the system, their registered RuleNodes are already known.

Further optimisations that may be implemented in future system embodiments are discussed in Chapter 13 (page 242).

11.8 Chapter Summary

This chapter has described some aspects of the detailed design of the 7Cs system and the FastFIX prototype implementation.

The 7Cs system is designed to capture the minimum set of knowledge required to identify and solve specific classes of customer problems. It allows multiple experts to concurrently and collaboratively modify the decision tree or decision mesh, share their differing expert opinions, and resolve their classification conflicts. Experts can refine the system's knowledge both top-down by editing and adding attributes and RuleNodes, and bottom-up by identifying key differences between conflicting cases. The 7Cs system allows users to record, retrieve, review, refine and rate trouble-shooting knowledge in the context of specific problem cases, and using existing solutions. Amongst other novel ideas, significant ideas included in the 7Cs system, implemented in the FastFIX prototype, and tested during the FastFIX software trial include:

1. The ability for multiple users to build an MCRDR-based decision tree in a hybrid top-down rule-driven and bottom-up case-driven wiki-style collaborative effort.
2. The ability for users to share their trouble-shooting questions and thereby help each-other to "work-up" their incoming cases using a novel Interactive and Recursive Multiple Classification RDR decision structure.
3. Continuous background monitoring of changes to the knowledge base via live versus registered case-RuleNode associations, so that users with affected RuleNodes and Cases can notice and respond to the changes. This approach allows classification conflicts to be identified, clarified and resolved in a manner not previously possible for RDR systems, and in so doing it increases the opportunities for and hence the speed of case-driven knowledge acquisition.

The creation, formation, development and testing of these novel ideas with real users in a real trouble-shooting environment has been one of the most significant contributions of this thesis.

The next chapter presents the results of the FastFIX software trial.