

N93-25980

Multi-Viewpoint Clustering Analysis¹

Mala Mehrotra

Vigyan Inc.

30 Research Drive

Hampton, VA 23666-1325

Chris Wild

Dept. of Computer Science

Old Dominion University

Norfolk, VA 23529-0162

Abstract

In this paper, we address the feasibility of partitioning rule-based systems into a number of meaningful units to enhance the comprehensibility, maintainability and reliability of expert systems software. Preliminary results have shown that *no single structuring principle or abstraction hierarchy is sufficient to understand complex knowledge bases*. We therefore propose the Multi-View Point - Clustering Analysis (MVP-CA) methodology to provide multiple views of the same expert system. We present the results of using this approach to partition a deployed knowledge-based system that navigates the Space Shuttle's entry. We also discuss the impact of this approach on verification and validation of knowledge-based systems.

Keywords domain knowledge, primary view, secondary view, conceptual clustering.

Introduction

Knowledge-based systems owe their appeal to the promise of utilizing expertise in the

¹This research was supported through Phase-I SBIR Grant - NAS9-18706 from NASA Johnson Space Center, Houston, TX.

domain knowledge for the solution of difficult, poorly-understood, ill-structured problems. However, they must be subjected to rigorous verification and validation (V&V) analyses before they can be accepted into real-world critical applications. Unfortunately, expert systems do not lend themselves to the traditional V&V techniques for highly reliable software. There is a need to formulate an acceptable set of V&V techniques which can assure their quality. Better knowledge-acquisition techniques as well as better management, understanding and enhancement of the knowledge base is critical to the success of such V&V activities.

The difficulty in the V&V of large knowledge-based systems arises due to a number of reasons. Firstly, rapid prototyping and iterative development form key features of any expert system development activity. This has led to the development of ad-hoc techniques for expert system design without any software engineering guidelines. Moreover, due to the data-driven nature of expert systems, as the number of rules of an expert system increase, the number of possible interactions between the rules increases exponentially. The complexity of each pattern in a rule compounds the problem of V&V even further. As a result, large expert systems tend to be incomprehensible,

difficult to debug or modify, and almost impossible to verify or validate.

Compounding the problem further is the fact that most expert systems are built without much regard to defining the requirements or specifications upfront. As any software, conventional or knowledge-based, becomes more complex, common errors are bound to occur through misunderstanding of specifications and requirements. Therefore, it is our belief that even if a software life cycle stresses specifications and requirements upfront, that will not be enough to guarantee the right product for complicated systems. There are bound to be ambiguities and interpretational problems. What is needed is a complementary tool that is capable of exposing such ambiguities and misinterpretations so that corrective action can be taken before it is too late in the software life cycle. Having a semi-automated means for capturing and structuring the meta-knowledge in a rulebase and cross-checking it with the specifications and requirements at various stages of the software life cycle could certainly help in this effort.

Conventional software yields more easily to verification efforts because control is explicitly represented as procedures which can be structured to encapsulate run-time abstractions. Modules can be designed in conventional software, each consisting of a manageable unit with a well-defined interface. Furthermore, procedures can be grouped into packages or objects which share an internal data structure. These units can then be subjected to unit/integration testing techniques.

Due to the declarative style of programming in knowledge-based systems, the generation of clusters to capture significant concepts in the domain seems more feasible than it would be for procedural software. By

using knowledge-based programming techniques one is much closer to the domain knowledge of the problem than with procedural languages. The control aspects of the problem are abstracted away into the inference engine (or alternatively, the control rules are explicitly declared). The existence of a model of the domain would benefit the analysis of other knowledge-based systems within that domain by providing seeds for cluster formation. In addition, the use of a domain model to assist in the development of new knowledge-based systems is a promising research direction.

Existing research indicates that misunderstandings of the domain are a primary cause of systems failures [5, 12, 19]. Often small oversights or misunderstood interactions between sources of expertise lead to catastrophic failures. Techniques, methodologies and supporting tools are therefore needed to manage a complex system from multiple viewpoints and discover subtle interrelating concepts that are so critical for assuring the reliability of these systems. Even though language support for systems structuring has long been recognized as a key aspect of modern software and knowledge engineering, it is our contention that *no single structuring can simultaneously capture all the important concepts in complex knowledge-based systems*. We believe that techniques, methodologies and supporting tools are needed to manage a complex system from multiple viewpoints and that the discovery of subtle interrelating concepts is critical for assuring the reliability of these systems.

In this paper, we propose the concept of Multi-Viewpoint Clustering Analysis (MVP-CA) and show it as a feasible and effective technique towards structuring a rulebase for capturing its explicit as well as its implicit knowledge. The extraction of implicit, previously unknown, yet potentially useful in-

formation from the rulebase can have considerable impact on various stages of the life cycle of knowledge-based systems software. It can expose various design pitfalls during construction of the rulebase and the functional limitations of the software during its operation, as well as the subtle interrelationships between subgroups of rules that could prove very valuable in the maintenance of the system. It is our contention that the understanding of any large knowledge base will require that it be viewed from several different, possibly orthogonal viewpoints. MVP-CA provides an ability to discover significant structures within the rulebase by providing a mechanism to structure both hierarchically (from detail to abstract) and orthogonally (from different perspectives). Moreover, transfer of expertise from one problem domain to another related domain would be facilitated through the factoring of common aspects across the domains. Hence software reuse can be exploited through multiple structuring of a knowledge-based system.

First, we give an overview of our approach, followed by the methodology used to generate meaningful partitions. Next, we present the results of applying this methodology to a deployed expert system for navigation. We discuss some of the related work in this area and finally give our conclusions.

MVP-CA Overview

Our research efforts address the feasibility of automating the identification of rule-groups in knowledge-based systems software, to reflect the underlying subdomains of the problem. We prove the feasibility of MVP-CA (Multi-Viewpoint Clustering Analysis) methodology by building an MVP-CA tool

to structure a few CLIPS² [3] knowledge-based systems along several viewpoints and showing that no single structuring principle or abstraction hierarchy is sufficient to understand complex knowledge bases.

Our approach utilizes clustering analysis techniques to group rules which share significant common properties and to identify the concepts which underlie these groups. Cluster analysis is a kind of unsupervised learning in which (a potentially large volume of) information is grouped into a (usually much smaller) set of clusters. If a simple description of the cluster is possible, then this description emphasizes critical features common to the cluster elements while suppressing irrelevant details. Thus, clustering has the potential to abstract from a large body of data, a set of underlying principles or concepts which organizes that data into meaningful classes. The knowledge acquisition process therefore involves "mining" the rule base for interesting concepts shared among the rules. The quality of clustering is related to two competing factors: intra-group cohesiveness and inter-group coupling. Informally, one can say that a group (or a cluster) is cohesive if all the items clustered together are somehow related or similar. Two groups are highly coupled if they share many similar properties and they are loosely coupled (possibly decoupled) if they share few (or no) similar properties. It is interesting to note that the qualities which define a good cluster are precisely those which define a good modular functional decomposition of a problem.

Preliminary experiments with the MVP-CA tool exposed significant natural structures within different knowledge bases. For example, consider ONAV (Onboard Navigation Expert System) [1], an expert system deployed on the shuttle to navigate dur-

²C Language Production System

ing re-entry. The file structure of ONAV provides one partitioning of the whole system. Not only did we find this generally accepted partitioning of ONAV, but we also found less obvious, more subtle interrelationships that existed across these primary clusterings. In this paper we present some of our results of applying the MVP-CA tool to ONAV. Misunderstandings of subtle interactions contribute most to the unreliability of knowledge-based systems [10]. Hence any methodology that exposes these relationships will contribute towards the V&V of large knowledge-based systems.

To illustrate the need for multiple viewpoints, consider an expert system for selecting the appropriate wine to complement a dinner. Even such a relatively small rule-base can be structured from several different viewpoints, as shown in Figure 1. Very broadly, the knowledge base can be divided into knowledge about the problem domain (selecting the appropriate wine) and knowledge about the control domain. The control knowledge breaks up further into user interface (how to question the user) and overall control strategies (balancing user preferences against experts' opinion through various phase control rules). Printout statements that ask the user for input or control the phasing of control rules belong to the control domain.

Similarly, knowledge about the problem domain, to aid in the selection of an appropriate wine for a meal, can be further subdivided into three major subdomains: types of food, wine properties and varieties, and a model of the customer's preferences. These domains are further subdivided into various subaspects. All these reflect different viewpoints of the same rule base. Within the food subdomain there are partitionings of taste of food, style of food, ingredients, etc. This is a hierarchical partitioning under

the food subdomain. An orthogonal viewpoint in the wine subdomain is the interaction of wine properties with meal qualities. Similarly there are different aspects of the problem from the customer's viewpoint. In addition, there are rules which overlap subdomains or pass information to rules in other subdomains (data dependency relationships). Thus the same rule can be part of one subdomain and at the same time create information for use by rules in other subdomains, such as interface rules that specifically combine concepts from two subdomains (e.g., the relationship between beverage and the style of food.) There is an added value in using the MVP-CA tool for exposing substructures within the abstract groups formed, through hierarchical partitionings generated by it. The hierarchies represent viewpoints at different levels of conceptual abstraction.

MVP-CA Methodology

The methodology used for MVP-CA is summarized graphically in Figure 2. In the *Cluster Generation Phase* the focus is on generating meaningful clusters through statistical and semantics-based measures. In the *Cluster Analysis Phase* the focus is on performing a statistical and functional analysis of the output generated from the previous phase. Results of a statistical analysis of the output data feed back as better constraints on the parameters for grouping to improve the quality of subsequent clusterings. A functional analysis of the clusters captures the key concepts conveyed by the clusters generated. *Concepts* are meaningful patterns in the rulebase along with their associated attributes. A set of key concepts constitutes a single *viewpoint*. Multiple *clusterings* present multiple viewpoints on the rule base.

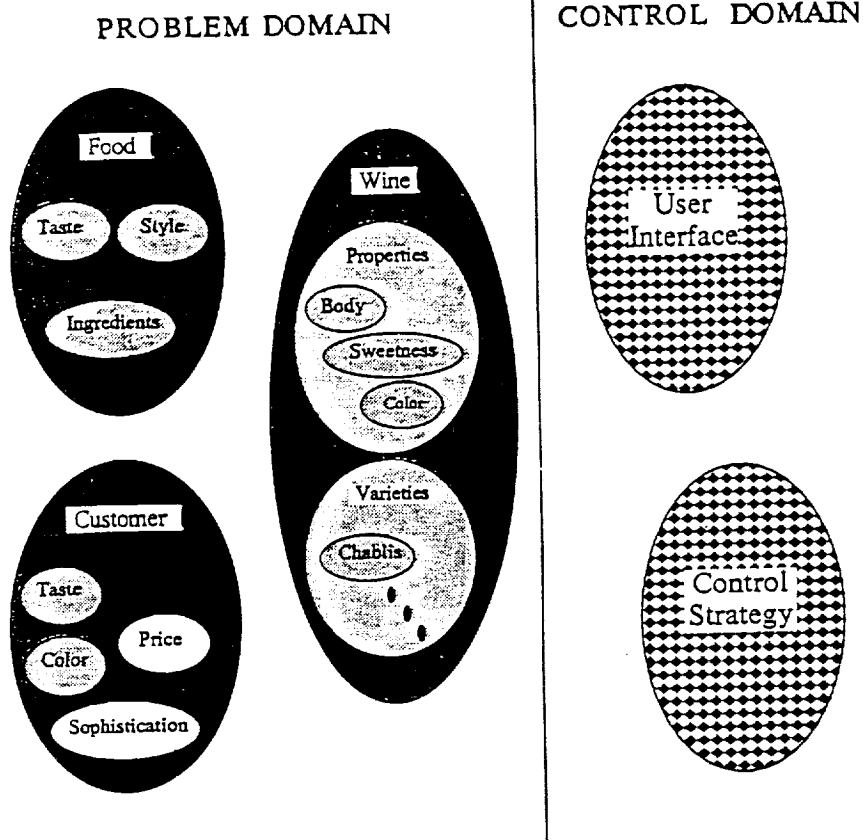
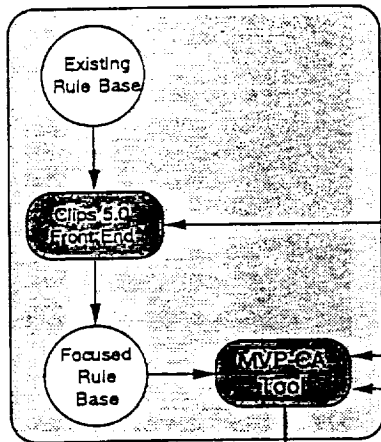
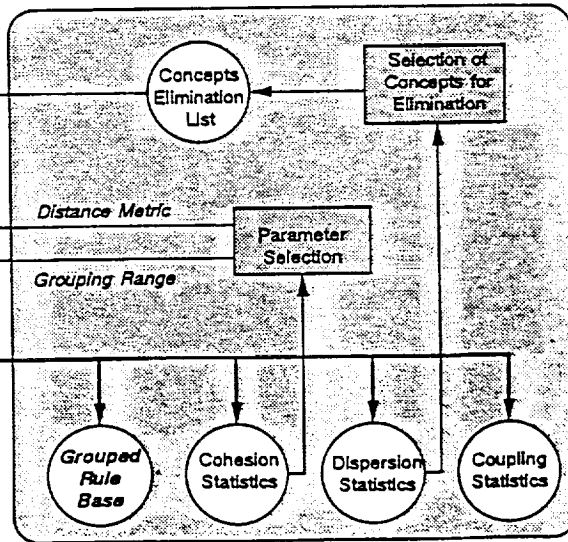


Figure 1: A Multi View Point of the Wine Rule Base

Cluster Generation Phase



Cluster Analysis Phase






-  Data File
-  Tool
-  User Input

Figure 2: Phase-I Data Flow Diagram

A two-step procedure is utilized for extracting multiple viewpoints of a rulebase. First, form the best cluster possible using various measures, such as dispersion, cohesion and coupling. The overall dispersion of a pattern p is

$$disp(p) = \sum_{i=1}^{n_C} disp_{G_i}(p)$$

where n_C is the number of groups for clustering C and $disp_{G_i}(p) = 1$ if $p \in G_i$ and is 0 otherwise. Coupling is defined in terms of the inter-group distance, $D(i, j)$ as follows:

$$D(i, j) = \sum_{r_k \in G_i} \sum_{r_l \in G_j} \frac{d(r_k, r_l)}{n_i * n_j}$$

where n_i and n_j are the number of rules in groups G_i and G_j , respectively and $d(r_k, r_l)$ is the distance between rules r_k and r_l defined according to a distance metric selected by taking into account the nature of the rule base application [15]. For a given clustering, C , the cohesiveness measure is an index of the similarity of rules belonging to the same group. Cohesiveness of a rule r_k with respect to the group G_i that it belongs to is the average number of concepts($cncp$) it shares with the other rule members in the group G_i .

$$coh_{G_i}(r_k) = \sum_{\substack{r_l \in G_i \\ (r_k \neq r_l)}} \frac{|2 * comm_cncp(r_k, r_l)|}{|cncp(r_k)| + |cncp(r_l)|}$$

Our clustering algorithm starts with all rules in their own clusters. At each step of the algorithm, the two groups which are most similar are merged together to form a new group. This pattern of mergings forms a hierarchical cluster from the single-member rule cluster to a cluster containing all the rules. One can look at this clustering near the "best" clustering points. Deciding which level in the hierarchy forms the "best" clustering of the rules requires an

analysis of the cohesiveness of each cluster (the intragroup similarity) versus the coupling between groups (the intergroup similarity). When group cohesiveness is plotted against number of groups, plateau regions are generated signifying stable values for cohesiveness in certain ranges of number of groups. These regions represent optimal partitionings for a particular level of conceptual abstraction. Insight into concepts dominating the various clusters can be obtained through an examination of the groups at select points on the plateau regions. A hierarchical view of the rulebase can then be generated by repeating the above procedure for different plateau regions on the cohesiveness plots.

Next, with this "best" cluster, form a concept focus list - to either sharpen a current viewpoint or expose an alternate viewpoint. The concept focus list is formed from dispersion statistics of patterns. Dispersion is based on shared concepts - i.e. how a single concept is dispersed among the clusters. Low dispersion concepts are likely to represent concepts which characterize the clusters they are in. In fact, high dispersion concepts may interfere with the generation of highly cohesive clusters. Removing these concepts before clustering can help define the clusters more distinctly - a process which we call "sharpening". However, high dispersion concepts may also represent legitimate alternate structurings of the knowledge base. By selectively removing the low dispersion concepts, it is possible to reveal subtle alternate viewpoints - a concept we have termed multi-viewpoint clustering analysis [17, 16]. Thus the MVP-CA methodology provides a mechanism for comprehending complex knowledge-based systems through structuring them both hierarchically (from detail to abstract) and orthogonally (from different perspectives) leading to discovery of signif-

icant structures within the rule base.

Experimental Results

In this section we present some of the results obtained to date with the deployed knowledge-based system ONAV. Other results using animal classification and wine selection (available as part of the CLIPS 5.1 release) expert systems have been presented in [16].

Even with extensive comments and a tool such as CRSV³ [2], the conceptual dependencies of rules across files cannot be easily determined. Not having any experience with Shuttle mission terminology, the rulenames were our only guide for understanding the domain in this knowledge-base. After clustering this rulebase several times using different criteria, we began to understand more of the subtle interrelationships. A graphical user interface, currently under development, would allow us to navigate through the rulebase and document the insights generated by the partitioning, thus fully utilizing the MVP-CA methodology. We document below our understanding of ONAV based on the natural partitionings set up by the developer as well as different groupings generated through the MVP-CA tool. We also show some of the interrelated concepts uncovered by this tool.

ONAV is an expert system developed at NASA Johnson to help navigate re-entry of a space craft. It has 387 rules divided across 16 files reflecting the various stages of navigation: ascent, entry and landing. The largest file *tacan.r* contains 127 rules. Monitoring of the space shuttle through ONAV entails updating some state vectors in the files *state.r*,

³CLIPS Cross Reference Style Analysis and Verification Tool

3state.r and *hstd.r*. Measurements of velocity and acceleration are calculated through sensor readings from various devices such as the inertial measurement unit (*imu*), drag unit(*drag*), barometer unit(*baro*), tactical air navigation unit(*tacan*) and microwave scan beam landing system(*msbls*). The readings go through a Kalman filter and the state vector is updated through different types of line replacement units (*lru*) attached to the different devices. The computers onboard perform the necessary integrations on the corrected readings to obtain accurate values of velocity and position.

During landing, readings from different sources have to be tallied so that the positioning of the shuttle can be as accurate as possible before it hits the runway. During ascent the shuttle relies mainly on the inertial measurement unit readings, since an accurate positional value is less critical. All the *lrus* feed data to both the primary avionics system software(PASS) as well as to the backup flight system(BFS). Each of these systems have different selection schemes for determining the quality of data received. Ground-based radar stations resolve any conflicting values for the position of the shuttle and are used to aid in isolating malfunctioning equipment on board. Fidelity of the data is monitored through the status of a number of different flags. Rules in *telemetry.r* and *operator.r* determine which of the readings and updated state vectors are reliable at any point in time and give the operator power to override any decision. *Tables.r* provides general information on the *lru* configurations onboard, the fault matrix to be used for identifying the *imu* component that has failed, and a definition of the quality ratings to be used for the different state vectors and data readings. Runway selections are checked out in the file *runway.r*. Rules in *init.r*, *control.r*, and *output.r* essen-

tially accomplish the initial set up of global information during the different stages of the navigation by activating the various phase control rules, and they also handle the user interface issues.

Initial analysis of our results indicates that grouping a rulebase according to control aspects of the problem is not sufficient for understanding the problem. The static aspects of the problem can be understood only if domain knowledge can be separated from control knowledge [8, 9]. The original partitioning of ONAV into 16 files by the developer provided only a coarse partitioning based on the different phase aspects of the knowledge-based system. When the phase aspects of the rulebase were excised, it was found that rules with similar domain information were formed into a single group to give a secondary view. In order to discover the implicit interconnections between rules in different files, we combined all the files of ONAV to form one 387-rule rulebase. Since ONAV is primarily a monitoring system with some diagnostic capabilities, more meaningful partitionings were obtained when the antecedent patterns played a major role in determining the distance between rules [15].

Figure 3 shows the cohesion plot for a primary view of ONAV. The cohesion values beyond 200 groups are not plotted because there are too many single groups after that point. Consider some of the interesting plateau regions such as those around 11 and 50 groups. Partitionings generated with the primary view are more or less in accordance with the developer's partitionings in the rulebase reflecting various phase values. At 50 groups, we can see various subaspects for the *tacan* subphase - such as, *tacan* prediction rules, rules that put *tacan* in automatic mode, rules to determine *lru* quality, and so on - grouped in separate groups. However, at 10 groups, all these

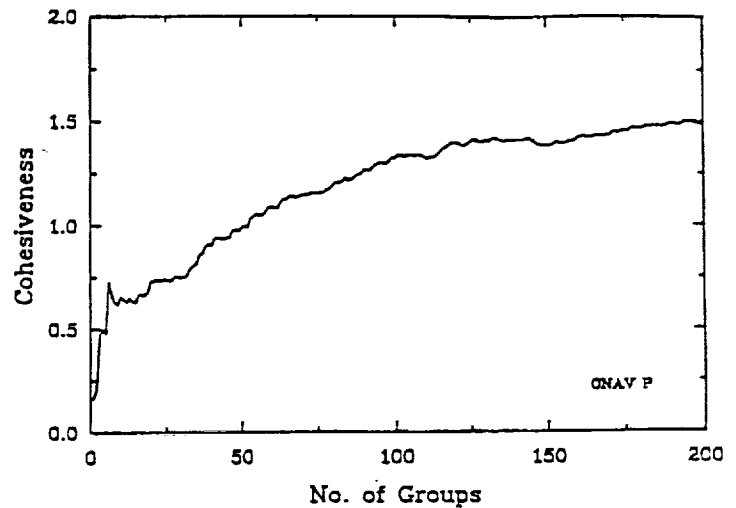


Figure 3: Cohesiveness Plot: ONAV rulebase - Primary View

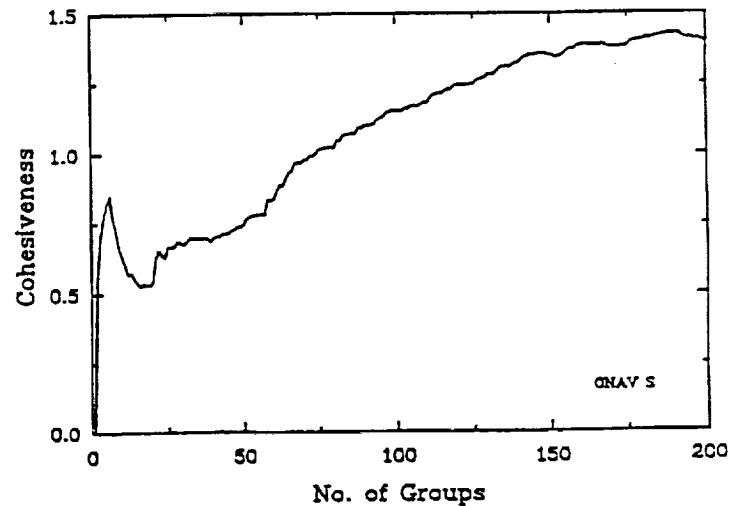


Figure 4: Cohesiveness Plot: ONAV rulebase - Secondary View


```

Group no 20:
Total number of rules in group: 15
Distance:: Min: 2.000000 Max: 7.666667 Mean: 4.284770
Cohesiveness: 0.429254 Minimum Membership: 0.033520
130   init-engaged-system-is-bfs      0.150933
131   init-engaged-system-is-pass     0.445672
134   init-system-availability-bfs-only 0.537089
135   init-system-availability-pass-only 0.566508
137   init-system-availability-both-pass-avail 0.561815
136   init-system-availability-both 0.565853
140   init-report-major-mode 0.451083
138   init-wrong-atmosphere 0.399324
139   init-right-atmosphere 0.371124
132   init-enable-msbls-sensor-lights 0.232097
133   init-enable-tacan-sensor-lights 0.295337
141   init-keep-last-ops-num 0.362514
142   init-report-abort-mode 0.501832
143   init-report-ascent-events 0.544941
223   nav-initialize 0.452685

```

Figure 5: Initialization Rules - Primary Clustering

tacan rules come together to form one group as conceived by the developer. Thus, while the original partitioning of ONAV into 16 files by the developer provided a coarse partitioning based on the different phase aspects of the knowledge-based system, there is added value in using the MVP-CA tool to expose the substructures within these abstract groups.

In the primary view, some groupings seem to have been generated based on criteria other than phase control. Initialization rules across different files come together in a group, group 20 in Figure 5, revealing initialization relationships from various phases. Initializations from other files, such as *nav-initialize* from file *state.r*, combine with this group revealing initialization relationships across files. This is an important revelation from the point of view of maintenance and verification.

In order to reveal a secondary view, we excised the concept of *phase* and *engaged-system*, which had the highest dispersion values in the primary view. The cohesion plot for the secondary view is given in Figure 4. Figures 7 and 8 give cross-sections of secondary groupings when all phase values were excised. The rule labelings generated in these files are the rulenames given by the developer originally. The numbers on the left are the rule numbers; distance between rule numbers thus gives an indication of the degree of juxtaposition of the rules in the combined rule base. Right-hand side numbers provide the cohesion value of the rule with respect to its group.

Once the phase aspect is deleted from the rulebase, other domain-dependent concepts start asserting themselves. In fact, in Figure 7, group 8 rules with similar rulenames (*hstd-same*, *hstd-bad*, *hstd-good* and *hstd-unavail*) across different files (*hstd.r* and *op-*

Group no 6:

Total number of rules in group: 19

Distance:: Min: 2.000000 Max: 6.000000 Mean: 3.688889

Cohesiveness: 0.443354 Minimum Membership: 0.160000

27	control-kickoff	0.385737
194	operator-stop	0.526758
201	operator-uplink-runway	0.454210
195	operator-delta-state	0.472484
196	operator-changed-delta-state	0.520917
197	operator-bfs-no-go	0.398486
198	operator-bfs-go	0.438764
199	operator-runway-selection	0.400035
200	operator-desired-runway-from-operator	0.443254
204	operator-atmosphere-change	0.375280
202	operator-toggle-tacan	0.342885
203	operator-cant-toggle	0.416190
205	gndeph-bad	0.443719
207	gndeph-same	0.490814
206	gndeph-good	0.452024
209	hstd-good	0.451576
208	hstd-bad	0.481044
210	hstd-same	0.534733
211	hstd-unavail	0.394810

Group no 12:

Total number of rules in group: 4

Distance:: Min: 2.333333 Max: 3.250000 Mean: 2.763889

Cohesiveness: 1.112825 Minimum Membership: 0.571429

42	hstd-bad	1.229437
44	hstd-same	0.884921
43	hstd-good	1.136364
45	hstd-unavail	1.200577

Figure 6: *Hstd* rules - Primary View

Group no 8:
 Total number of rules in group: 24
 Distance:: Min: 2.000000 Max: 9.900000 Mean: 4.437921
 Cohesiveness: 0.377193 Minimum Membership: 0.000000

27	control-kickoff	0.351796
211	hstd-unavail	0.353633
194	operator-stop	0.479850
201	operator-uplink-runway	0.414068
210	hstd-same	0.492559
195	operator-delta-state	0.459732
196	operator-changed-delta-state	0.487993
197	operator-bfs-no-go	0.375855
198	operator-bfs-go	0.405531
199	operator-runway-selection	0.355490
200	operator-desired-runway-from-operator	0.391028
205	gndeph-bad	0.440268
207	gndeph-same	0.446688
202	operator-toggle-tacan	0.318074
203	operator-cant-toggle	0.389968
43	hstd-good	0.294328
206	gndeph-good	0.444678
209	hstd-good	0.472972
42	hstd-bad	0.316276
208	hstd-bad	0.487546
44	hstd-same	0.220726
138	init-wrong-atmosphere	0.090802
139	init-right-atmosphere	0.148827
204	operator-atmosphere-change	0.413935

Figure 7: *Hstd* rules - Secondary View

Group no 5:
 Total number of rules in group: 4
 Distance:: Min: 2.000000 Max: 4.000000 Mean: 3.000000
 Cohesiveness: 1.328788 Minimum Membership: 0.013423

20	baro-aif-changed	1.176493
36	drag-aif-changed	1.653500
310	tacan-aif-changed	1.372859
179	msbls-aif-changed	1.112300

Figure 8: *Aif* rules - Secondary View

erator.r) come together because all of these rules deal with an incorrect input value for the *hstd* indicator. However, the *hstd* indicator is important in two subphases (*fact-assertion* and *hstd*). Once the phase component is deleted, the domain information that determines the *hstd* status pulls these rules into the same group. In the primary view these rules were in separate groups, 6 and 12, as shown in Figure 6.

It is also interesting to note that rules that share the concept of modifying the auto-inhibit-force flag (*aif*) in different phases all combine together in group 5, see Figure 8. This is a functional grouping of rules based on actions to be taken when there is a discrepancy between the previous and current values of the *aif* flag in the *barometer*, *drag*, *tacan* and *msbls* units. An orthogonal view of the rulebase comes into perspective with this grouping.

Such a view may be of immense value to the maintainer of the rulebase, since functional dependencies like these can be extremely difficult to locate across files, especially if the maintainer has not been the original developer of the system. Thus, our experimental results with the MVP-CA tool has demonstrated the feasibility of discovering significant structures within the rulebase by providing a mechanism to structure both hierarchically (from detail to abstract) and orthogonally (from different perspectives).

Related Work

Extraction of meta-knowledge for the purposes of comprehending and maintaining expert systems has been an accepted norm. In this section, we examine the role of structuring for this purpose in some well-established knowledge-based systems.

Systems such as XCON [4, 18] that have been in development for more than 10 years had to develop a new rule-based language, RIME, and rewrite XCON-in-RIME to facilitate its maintenance. XCON-in-RIME is supposed to make the domain knowledge more explicit both in terms of restructuring the rules and in terms of exposing the control structure for firing of the rules. Thus the problem space gets more hierarchically organized into different functional aspects, the problem solving method is made more explicit, a domain-specific classification is imposed on the rules and rule templates are created to serve as guides for rule creation.

Meta-Dendral [6] is a case study in the area of acquisition of domain knowledge. Meta-Dendral tries to resolve the bottleneck of knowledge acquisition through automatic generation of rule sets so as to aid the process of formation of newer scientific theories in mass spectroscopy.

TEIRESIAS [7] is built upon the MYCIN system to provide a mechanism for effective knowledge transfer. TEIRESIAS uses meta-rules to encode rule-based strategies that govern the usage of other rules. For this purpose it generates a set of rule models that are then used to guide this effort by being suggestive of both the content and form of the rules. These rule models can suggest incomplete areas of the knowledge base, provide summary explanations and help during debugging sessions. TEIRESIAS demonstrates the power of analyzing rule sets for experts especially when writing new rules. It is very helpful to see existing rules that are similar to a new rule under consideration so as to set the appropriate certainty factors in the new rule. Similarity could be suggestive of similar premises or similar conclusions. By comparing other evidence and other conclusions, the strength of the proposed rule can be estimated in the proper context. In fact,

each of the clusterings carries an extra slot indicating the context in which the rule set applies.

Although others [11, 13, 14] have attempted to cluster knowledge bases in order to abstract and structure the knowledge in them, existing approaches are limited in two major ways. First, we believe that *no one single structuring viewpoint is sufficient to comprehend a complex knowledge base*. Second, it is difficult to understand a single knowledge base isolated from an understanding of the underlying application domain. Often clues to the underlying semantic concepts are provided through descriptive names. Even then, the syntactic structure alone is rarely sufficient for managing and maintaining a complex system.

Clustering analysis can be used to reveal regularities in the knowledge base which can suggest possible subdomains of the problem. This structuring of the knowledge base is intended to capture both the explicit and the implicit knowledge in the knowledge base. *The point of interest of such an analysis should not be the clusters themselves, but the principles and ideas suggested by the clusters*. Such groups would allow one to abstract away from the point of view that *each rule is a procedure call* and look at the system from higher semantic levels. Each such group or unit can then be viewed as a procedure having a well-defined interface to other rule-groups. Once a rule base is decomposed into such "firewalled" units, studying the interactions between rules would become more tractable.

Due to the declarative style of programming in knowledge-based systems, the generation of clusters to capture significant concepts in the domain seems more feasible than it would be for procedural software. By using knowledge-based programming tech-

niques one is much closer to the domain knowledge of the problem than with procedural languages. The control aspects of the problem are abstracted away into the inference engine (or alternatively, the control rules are explicitly declared.) Generation of a model of the problem domain can be accomplished through clustering. The existence of a model of the domain would benefit the analysis of other knowledge-based systems within that domain by providing seeds for cluster formation. In addition, the use of a domain model to assist in the development of new knowledge-based systems is a promising research direction.

Conclusions

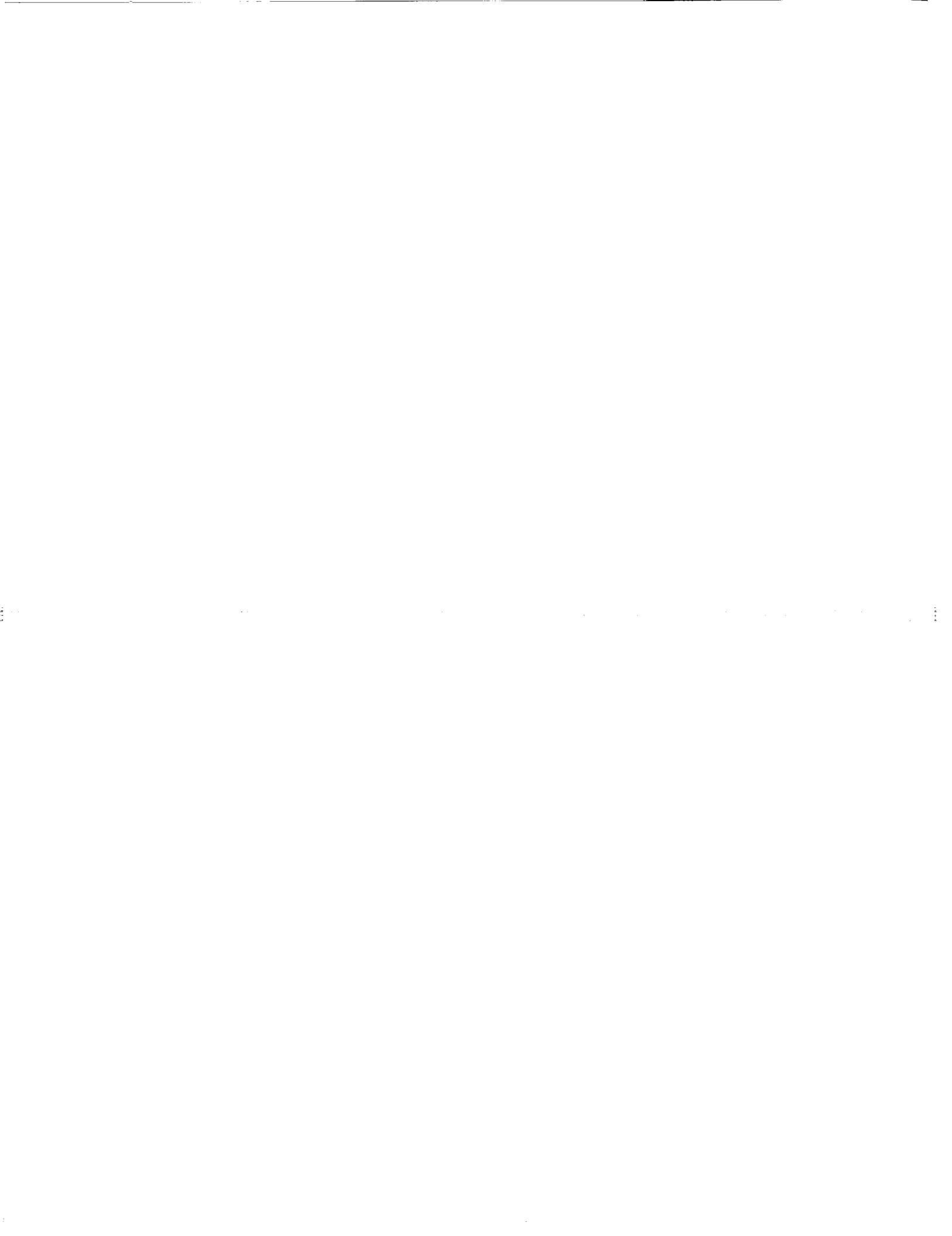
Knowledge-based systems have the potential to greatly increase the capabilities of many aerospace applications such as Space Station, manned and unmanned spacecraft and civilian and military air transport. Automated systems that are knowledge based need to be deployed aboard these missions to reduce manpower support. Failure of such systems, however, can result in loss of life and of substantial financial investment. Hence these systems need to be highly reliable. Whereas DOD standards for conventional software have been developed, such as ADA-9x, a credible development and validation methodology for knowledge-based systems is currently lacking. Acceptance of knowledge-based systems software for critical missions is very much dependent on development of effective software engineering and validation techniques. A structured approach to management and maintenance of such systems would go a long way towards dispelling the myth that expert systems are inherently unreliable and that nothing can be done about it.

Expert systems have a wide commercial applicability. Liability issues arising out of improper functioning of such systems demand that any risk to life or property be either totally eliminated or at least minimized. Hence, it is imperative to develop rigorous and automatic testing tools for the verification and validation of knowledge-based systems. An integrated environment for expert system verification and validation, such as is proposed by MVP-CA, would overcome this barrier, opening them up for a broad range of important applications. An integrated system for performing V&V on structured knowledge bases will enhance the reliability of knowledge-based software and bridge its current gap with conventional systems.

References

- [1] Knowledge Requirements for the Onboard Navigation Console Expert/Trainer System. Technical Report JSC-22657, NASA, Lyndon B. Johnson Space Center, Houston, TX., September 1988.
- [2] CLIPS Reference Manual. Technical Report JSC-22948, Artificial Intelligence Center, NASA, Lyndon B. Johnson Space Center, Houston, TX., June 1989.
- [3] *CLIPS Basic Programming Guide CLIPS Version 5.1*. Houston, TX., September 1991.
- [4] V. E. Barker and D. E. O'Connor. Expert Systems for Configuration at Digital: XCON and beyond. *Communications of the ACM*, 32(3), March 1989.
- [5] V. R. Basili and B. T. Perricone. Software Errors and Complexity: An Empirical Investigation. *Communications of the ACM*, 1(27):42-52, January 1984.
- [6] B.G. Buchanan. Issues of Representation in Conveying the Scope and Limitations of Intelligent Assistant Programs. In J.E. Hayes, D. Michie, and L.I. Mikulich, editors, *Machine Intelligence*, pages 407-425. John Wiley & Sons, 1979.
- [7] B.G. Buchanan and E.H. Shortliffe. Knowledge Engineering. In *Rule-Based Expert Systems*, chapter 7, pages 149-158. Addison Wesley Publishing Co., 1985.
- [8] B. Chandrasekharan. Generic tasks in knowledge-based reasoning: High-level building blocks for expert systems design. *IEEE Expert*, Fall 1986.
- [9] W. J. Clancey. The advantages of abstract control knowledge in expert system design. In *Proceedings, National Conference on Artificial Intelligence*, pages 74-78, 1983.
- [10] D. Hamilton and K. Kelley. State-of-the-practice in knowledge-based system verification and validation. *Expert Systems with Applications*, 3:403-410, 1991.
- [11] R. J. K. Jacob and J. N. Froscher. A Software Engineering Methodology for Rule-based Systems. *IEEE Transactions on Knowledge and Data Engineering*, 1990, in press.
- [12] N. Leveson. Software Safety: What, Why and How. *Computing Surveys*, 2(18):125-164, June 1986.
- [13] S. Lindell. Keyword Cluster Algorithm for Expert System Rule Bases. Technical Report SD-TR-87-36, The Aerospace Corporation, El Segundo, CA., June 1987.
- [14] K. Lindenmayer, S. Vick, and D. Rosenthal. Maintaining an Expert System for the Hubble Space Telescope Ground Support. In *Proceedings, Goddard Conference on Space Applications of Artificial Intelligence and Robotics*, pages 1-13, May 1987.
- [15] M. Mehrotra. Rule Groupings: A Software Engineering Approach towards Ver-

- ification of Expert Systems. Technical Report NASA CR-4372, NASA Langley Research Center, Hampton, VA., May 1991.
- [16] M. Mehrotra, C. Wild, and D. Rosca. A Software Engineering Approach Toward Validation of Knowledge-Based Systems. Technical report, ViGYAN SBIR Phase-I Final Report, Hampton, VA., August 1992.
- [17] M. Mehrotra, C. Wild, and D. Rosca. Role of clustering analysis in the verification of expert systems. In *Notes for the AAAI-92 Workshop on Verification, Validation and Testing of Knowledge-Based Systems*, July 1992.
- [18] E. Soloway, J. Bachant, and K. Jensen. Assessing the Maintainability of XCON-in-RIME: Coping with the Problems of a VERY Large Rulebase. In *Proceedings of AAAI-87*, July 1987.
- [19] C. Wild, J. Chen, and D. Eckhardt. Reasoning about Software Specifications: A Case Study. *Proceedings of AIAA Computers in Aerospace VII Conference*, pages 297-306, October 1989.



Information Management

