Maintaining an Expert System for the Hubble Space Telescope Ground Support

May 13, 1987

Kelly Lindenmayer [1]
Astronomy Programs, Computer Sciences Corporation

Shon Vick
Space Telescope Science Institute [2]
3700 San Martin Drive
Baltimore, MD 21218

and
Don Rosenthal
NASA Ames Research Center

Abstract

The Transformation portion of the HST Proposal Entry Processor System converts an astronomer-oriented description of a scientific observing program into a detailed description of the parameters needed for planning and scheduling. The Transformation system is one of a very few rulebased experts systems that has ever entered an operational phase. The day to day operation of the system and its rulebase are no longer the responsibility of the original developer. As a result, software engineering properties of the rulebased approach become more important. In this paper, we discuss maintenance issues associated with the coupling of rules within a rulebased system and offer a method for partitioning a rulebase so that the amount of knowledge needed to modify the rulebase is minimized. This method is also used to develop a measure of the coupling strength of the rulebase.

[1] Staff Member of the Space Telescope Science Institute

[2] Operated by the Association of Universities for Research in Astronomy for the National Aeronautics and Space Administration

1 Introduction

The Hubble Space Telescope (HST) is an orbiting optical observatory to be launched by the Space Shuttle in late 1988. Using a rulebased expert system written in OPS5, the Transformation system converts an astronomer-oriented description of a scientific observing program into a detailed description of the parameters needed by the planning and scheduling portion of the HST Science Operations Ground Support System.[3] Transformation has been in an operational phase since December 1985. During its early stages of the operational phase, the primary designer and implementer of the rulebased portion of the Transformation System remained responsible for development and maintenance of the rulebase. Eight months later, the system was turned over to a member of the software group who had limited prior exposure to the project. The Transformation system is one of a very small number of rulebased expert systems that has entered an operational phase. Rulebased systems have traditionally been developed as research projects, and have been maintained by their original implementers. As expert system techniques and technolgy mature, a trend towards developing rulebased systems for practical applications will occur. These systems will be developed with the expectation that they will grow and evolve throughout the life of the project. With this evolution, the notion of a software life cycle becomes relevant and the software life cycle that rulebase systems undergo does not correspond to that of conventional software systems.[4] Unlike software problems that are solved by a more algorithmic approach, AI programs tend to implement problems which have not been completely specified. As a result, expert systems raise some interesting software engineering considerations. In this paper, we discuss some of the maintenance issues associated with the coupling of rules for a rulebased system. We present a method for partitioning a rulebase in such a way that the knowledge required to make a modification to the rulebase is minimized. The method is also used to derive a metric that can be used as a measure of coupling strength.

2 Background

In OPS5, like many expert system languages, knowledge is encoded as a set of rules or productions with the following format:

IF antecedents THEN consequents

Facts from a global database, usually termed working memory, are matched with the antecedents (also referred to as left hand side or LHS). The matched rule/facts pairs (instantiations) are put into a conflict set. A conflict resolution strategy is applied to the set to determine the instantiation to be *fired.* During the *firing process*, the consequent prescribed by the rule corresponding to the instantiation is established. In general, this process creates new facts, and the inference cycle proceeds to the match step. This process continues until the conflict set is empty or the process is halted explicitly by the programmer. This paradigm encourages the *opportunistic* behavior of rules and when the situation is right (e.g. when the antecedents match the facts), the rules are instantiated and fired.

[3] Transformation is described in detail in [7]

[4] See [5] for more information

This phenomenon distinguishes rulebase systems from an algorithmic approach to software problems. In other words, the order of application of the program's functional pieces (in this case rules), is not built into the program, and will not be determined until runtime.

On another level, such inferencing systems may seem to be common coupled. In common coupling, functional modules are linked through a global data structure. In OPS5 any rule may potentially read or write any part of working memory since rules share the entire working memory data area. However, working memory is partitioned into elements of data classes (analogous to traditional data structures), and rules refer to working memory elements by class name, and attribute (field) name. Therefore, coupling in rulebased systems more closely resembles stamp coupling. Stamp coupling is similar to common coupling except that the global data items are shared more selectively between components that use them.[5]

The partitioning of groups of rules according to a goal strategy is presently one of the more viable method of controlling inferencing in OPS5 (compared to, for example running several smaller ruebased programs in sequence). There is no way for the developer to "program" the conflict resolution strategy, she may only choose between MEA and LEX.[6] Meta rules are not presently possible due to the strict partitioning of rule memory, working memory, and the conflict set.[7]

The Transformation rulebase is a goal oriented rulebase which consists of seven goals, some of which contain sub goals and task lists, but all control is imposed at a very high level. The rulebase is "partitioned" into goal sets which reside in files; that is, rules pertaining to a specific goal or subgoal are found in a single file. When writing rules, the natural tendency is to group rules according to their functionality. For instance, a knowledge engineer might group in a file all rules that merge exposures. From the maintainers standpoint, this method has its disadvantages as well as advantages. The most obvious advantage is that if a new rule to merge exposures needs to be added, there are several examples for a maintainer to follow. In addition, if a problem arises in how exposures are merged, the problem area may be localized, and hence easier to debug. The disadvantage lies in what occurs downstream from this problem. For instance, if a new rule is added to merge exposures, what else might that change affect? This is a formidable problem for a maintenance programmer who is not well versed in the structure inherent in the rulebase.

Other AI developers [8] have suggested an automated approach to partitioning a rule set based on a measure of "relatedness". Rules would be grouped according to the facts that they share - where a fact is some data representation that if changed in one rule would affect another rule in some way. Since there are several ways in which two rules could reference the same fact, the facts would be weighted, and the measure of "relatedness" would be based on these facts. The rulebase would then be partitioned into groups according to how strongly rules were related. Although these arguments and the related tools have merit, there are still some issues that need to be addressed. For instance, choosing a measure of relatedness is still a rather arbitrary process, although a more sophisticated clustering algorithm to partition the rulebase might be helpful. In addition, within a partitioned rulebase, there

[5] See [4] and [2] for a complete discussion of common and stamp coupling

[6] Refer to [1] for an explanation of MEA and LEX

[7] See [6] for a discussion of meta rules

[8] See [3] for details

may be subtle interactions between rules within a set of grouped rules or between the groups themselves. Moreover, the grouping procedures do not guarantee a small number of groups. What is needed then, is a method for maintenance programmers to understand the coupling of rules in general.

3 A Model for Partitioning a Rulebase

We first take a look at the possible ways in which rules may be coupled. We know that rules are related to one another through the facts that they share. In OPS5 there are essentially three different ways of altering facts in working memory: through a make, modify or remove. A *make* creates a new working memory element. A *remove* deletes a working memory element from the database. Conceptually, a *modify* is a combination of a make and a remove. It deletes a working memory element, and replaces it with the appropriate new working memory element.

Below is an example of two rule which are coupled by a *make* action. (All of the following examples are actual rules from the Transformation rulebase. For the purposes of understanding how rules are coupled, it is not necessary to comprehend the semantic content of the rule.) The first rule, *Find-parallel-with-mergeable-exposures* uses the make command to create a working memory element with class name *mergeable-exposures*. Since the specified attributes of this working memory element "match" a subset of the antecedent of rule *Remove-mergeable-exposures-if-same-mode-diff-aperture*, we consider these rules to be related.

```
(p find-parallel-with-mergeable-exposures

   (goal
       ^has-name        merge-exposures
       ^has-status      active
       ^task-list       find-potential-exposure-merges)

   (exposure-link
       ^has-exposure-number          <parallel-exposure>
       ^is-linked-to                 <primary-exposure>
       ^has-link-type                parallel-with )

   (mergeable-level
       ^symbol                       parallel-with
       ^value                        <parallel-with-level>)
-->

(make  mergeable-exposures
       ^first-exposure-number        <primary-exposure>
       ^second-exposure-number       <parallel-exposure>
       ^is-unmergeable               false
       ^is-mergeable-level           <parallel-with-level>
       ^merge-type                   parallel-with
       ^has-unique-label             (genatom) ) )
```

```
(p Remove-mergeable-exposures-if-same-mode-diff-aperture

   (goal
       ^has-name          merge-exposures
       ^has-status        active
       ^task-list         find-potential-exposure-merges)

   (exposure-specification
       ^is-internal-target-type      <> I
       ^has-exposure-number          <first-exposure>
       ^first-aperture-used          <aperture>

   (mergeable-level
       ^symbol                       parallel-with
       ^value                        <parallel-with-level> )

   {<mergeable-exposure-link>
   (mergeable-exposures
       ^first-exposure-number      <first-exposure>
       ^is-mergeable-level         <parallel-with-level>
       ^second-exposure-number     <second-exposure> ) }

   (exposure-specification
       ^has-exposure-number        <second-exposure>
       ^is-internal-target-type    <> I
       ^first-aperture-used        <> <aperture>
-->

   (modify <mergeable-exposure-link>
       ^1 unmerged-exposures))
```

When discussing relatedness, it is important to keep in mind that this is a static analysis of the problem. Because rulebased systems are data driven, rules will interact differently with different sets of data. A static approach to the problem is simply, what portion of the rulebase might *possibly* be affected by a particular rule change, given any set of data.

Figure 1 gives a pictorial representation of the relationship between these two rules. It shows that if *Find-parallel-with-mergeable-exposures*were modified, there is a *possibility* that *Remove-mergeable-exposures-if-same-mode-diff-aperture* might be affected by the change.

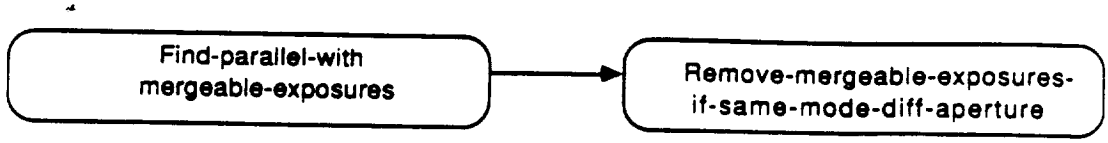

Figure 1.0 *Coupling Rules through a make action*

Next we look at two rules that are coupled by a *remove* action. The rule, *Remove-mergeable-exposures-if-first-is-an-acquisition* removes a working memory element that may have been created by *Find-parallel-with-mergeable-exposures*:

```
(p find-parallel-with-mergeable-exposures

   (goal
       ^has-name        merge-exposures
       ^has-status      active
       ^task-list       find-potential-exposure-merges)

   (exposure-link
       ^has-exposure-number          <parallel-exposure>
       ^is-linked-to                 <primary-exposure>
       ^has-link-type                parallel-with )

   (mergeable-level
       ^symbol                       parallel-with
       ^value                        <parallel-with-level>)
-->

(make  mergeable-exposures
       ^first-exposure-number        <primary-exposure>
       ^second-exposure-number       <parallel-exposure>
       ^is-unmergeable               false
       ^is-mergeable-level           <parallel-with-level>
       ^merge-type                   parallel-with
       ^has-unique-label             (genatom) ) )


(p Remove-mergeable-exposures-if-first-is-an-acquisition)

   (goal
       ^has-name          merge-exposures
       ^has-status        active
       ^task-list         find-potential-exposure-merges)

  {<mergeable-exposure-link>
   (mergeable-exposures
       ^first-exposure-number       <first-exposure> ) }

   (exposure-link
       ^has-exposure-number         <first-exposure>
       ^has-link-type               <<ONBOARD-ACQ INT-ACQ > )
-->

(remove <mergeable-exposure-link>) )
```

We will represent the relationship of two rules coupled by a *remove* as is shown in figure 2.0. Note that we picture the relationship slightly different for a *remove* coupling than we did for a *make* coupling. In the case of a *remove* , the arrow is pointing in the

opposite direction to show that in order for *Find-parallel-with-mergeable-exposures* to be affected by a modification of the rule *Remove-mergeable-exposures-if-first-is-an-acquisition*, *Find-parallel-with-mergeable-exposures* must already have been instantiated based on this working memory element.

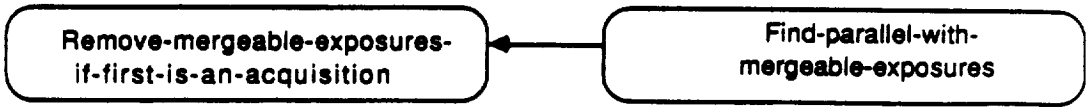

Figure 2.0 *Coupling Rules through a remove action*

In order to demonstrate the coupling created by a *modify*, it is necessary to use three rules. The first rule *Remove-fos-hrs-merge-if-only-second-mode-is-rapid* uses the *modify* action to change the class name of *mergeable-exposures* to *unmerged-exposures*. Since the specified attributes of the working memory element with class name *unmerged-exposures* "matches" the antecedent of the rule *link-alignments-with-unmerged-exposures*, these rules are said to be coupled. On the other hand, when *Remove-fos-hrs-merge-if-only-mode-is-rapid* uses the *modify* action to change the class name of a working memory element it is in affect removing the working memory element *mergeable-exposures*. So, based on the same principle as the *remove* example, we consider these two rules also to be coupled.

```
(p Find-same-orientation-mergeable-exposures

   (goal
       ^has-name           merge-exposures
       ^has-status         active
       ^task-list          find-potential-exposure-merges )

   (exposure-link
       ^has-exposure-number          <linked-exposure>
       ^is-linked-to                 <main-exposure>
       ^has-link-type                SAME-ORIENT )

   (exposure-specification
       ^has-exposure-number          <main-exposure>
       ^uses-SI-configuration        <SI-configuration> )

   (exposure-specification
       ^has-exposure-number          <linked-exposure>
```

```
        ^uses-SI-configuration          <SI-configuration> )

    (mergeable-level
        ^symbol                         same-orientation
        ^value                          <same-orientation-level>)
-->

(make   mergeable-exposures
        ^first-exposure-number          <main-exposure>
        ^second-exposure-number         <linked-exposure>
        ^is-unmergeable                 true
        ^is-mergeable-level             <same-orientation-level>
        ^merge-type                     same-orientation
        ^has-unique-label               (genatom) ) )


(p Remove-fos-hrs-merge-if-only-second-mode-is-rapid

    (goal
        ^has-name           merge-exposures
        ^has-status         active
        ^task-list          find-potential-exposure-merges )

    {<mergeable-exposure-link>
     (mergeable-exposures
        ^first-exposure-number          <first-exposure>
        ^second-exposure-number         <second-exposure>
        ^merge-type                     <<sequence-no-gap consecutive
                                          same-orientation>>
        ^is-unmergeable                 true) }

     (exposure-specification
        ^has-exposure-number            <first-exposure>
        ^uses-SI-configuration          << fos/bl fos/rd hrs >>
        ^uses-SI-operating-mode         <> rapid )

    (exposure-specification
        ^has-exposure-number            <second-exposure>
        ^uses-SI-operating-mode         rapid )
-->

(modify <mergeable-exposure-link>
        ^1          unmerged-exposures ))



(p link-alignments-with-unmerged-exposures

    (goal
        ^has-name merge-alignments
        ^has-status active
        ^task-list find-potential-alignment-merges)
```

```
    (unmerged-exposures
        ^first-exposure-number <first-exposure-number>
        ^first-copy-number <first-copy>
        ^second-exposure-number <second-exposure-number>
        ^second-copy-number <second-copy> )

    (assignment-record
        ^has-Pepsi-exposure-number <first-exposure-number>
        ^is-assignment-record-copy-number <first-copy>
        ^has-alignment-order <first-alignment-order>
                                                                    )

    (assignment-record
        ^has-Pepsi-exposure-number <second-exposure-number>
        ^is-assignment-record-copy-number <second-copy>
        ^has-alignment-order { <second-alignment-order> <>
          <first-alignment-order> }
                                                                        )
-->

(make mergeable-alignments
        ^has-first-alignment-order <first-alignment-order>
        ^has-second-alignment-order <second-alignment-order>
        ^has-unique-label (genatom) ) )
```

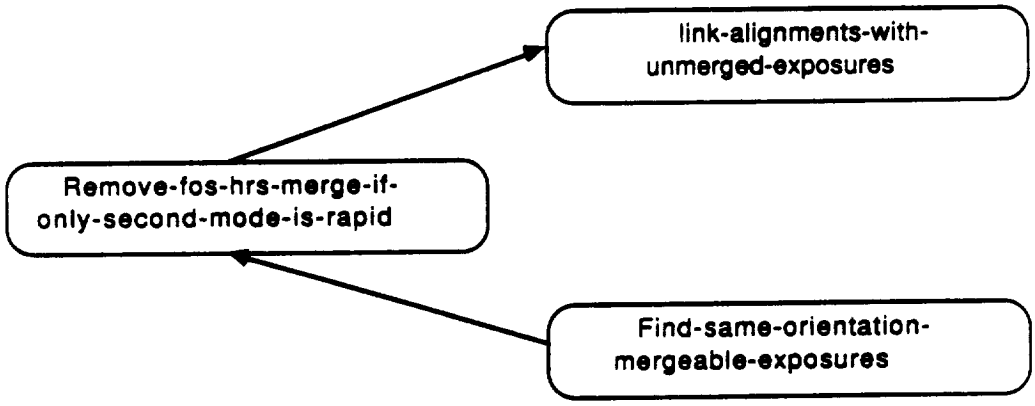

Figure 3.0 *Coupling Rules through a modify action*

4 A More Formal Way to Express Coupling

What we have seen so far is that if two rules are coupled by our definition, the **rhs** of one rule *feeds* the **lhs** of another rule. But what is actually meant by one rule *feeding* another? Let's introduce a notation to make the notion precise.

We can think of a rule as consisting of three parts: its name, its associated conditions, and its actions. A rule j is denoted by R_j.

$$R_j((c_{j,1} \ldots c_{j,n}) \rightarrow (a_{j,1} \ldots a_{j,m}))$$

The first condition of the **lhs** of R_j is called $c_{j,1}$, the second $c_{j,2}$ and so on. Similarly the first action will be $a_{j,1}$ and so forth.

We say R_j feeds R_i if the pattern corresponding to some action of R_j , say $a_{j,x}$ matches some condition clause of rule R_i, say $c_{i,y}$. This is expressed in a predicate calculus as :

$$\exists\, x\, \exists\, y\ \ (p_m\ (f_p\ a_{j,x}\ c_{i,y}))$$

where f_p is a function that given some rule action returns the pattern to which the rule action corresponds, and p_m is a predicate that takes two patterns and returns true if they match and false otherwise.

As examples of the notation suppose R_j is the rule *find-parallel-with-mergable-exposures* and R_i is the rule *Remove-mergable-exposures-if-first-is-an-acquisition*. Then:

$(f_p\ a_{j,1}) =$

```
(mergeable-exposures
    ^first-proposal-id          <parallel-proposal-id>
    ^first-version              <parallel-version>
    ^first-exposure-number      <primary-exposure>
    ^second-proposal-id         <parallel-proposal-id>
    ^second-version             <parallel-version>
    ^second-exposure-number     <parallel-exposure>
    ^is-unmergeable             false
    ^is-mergeable-level         <parallel-with-level>
    ^merge-type                 parallel-with
    ^has-unique-label           (genatom)
```

and

$c_{i,2}$ is

```
(mergeable-exposures
    ^first-proposal-id          <proposal-id>
    ^first-version              <version>
    ^first-exposure-number      <first-exposure> )
```

and $(p_m\ (f_p\ a_{j,1})\ c_{i,2})$ is *true*.

The general operation of the predicate p_m on the simple type matches should now be presented. If the element class of two patterns differs, then the predicate returns false. Otherwise, an attribute by attribute match is attempted. If an attribute is paired with a variable in one or both patterns, then the patterns match on that attribute. If they both are paired with the same constant, then again they match. If the attribute is not mentioned in the condition pattern, then it matches the action pattern for the attribute. Note that the p_m predicate returns true if all corresponding pairs of attributes match.

5 Using the Network

Clearly if every rule is coupled with every other rule in the system then the system is completely coupled. The maximum number of arcs is then n^2, where n is the number of rules in the rulebase. The ratio of actual arcs in the network to n^2 is a measure of the degree to which the rulebase is coupled. If the ratio is unity then any rule could interfere with any other. The lower the ratio the more local is the effect of a typical rule in the rulebase and the higher the degree of stamp coupling in the system.

To find the group of rules whose behavior **may** be directly affected by the addition of a new rule we need only regard the network of rules. Figure 4.0 represents a part of the network for some rulebase. The dotted arcs in the figure represent the new coupling that will occur if rule *new-remove* is added to the rulebase. Suppose for the sake of simplicity that both R1 and R6 contain a single make action on their **rhs**. The solid coupling lines flowing from R1 represent the match that occurs between the newly created wme from the make action of R1 and the conditional elements on the **lhs** of R2 and R3. (This is also true for the solid lines between R6 and R3,R4, and R5). The dotted line from R1 to *new-remove* shows that the make-pattern which creates a working memory element will match some set of conditional elements on the LHS of *new-remove*.

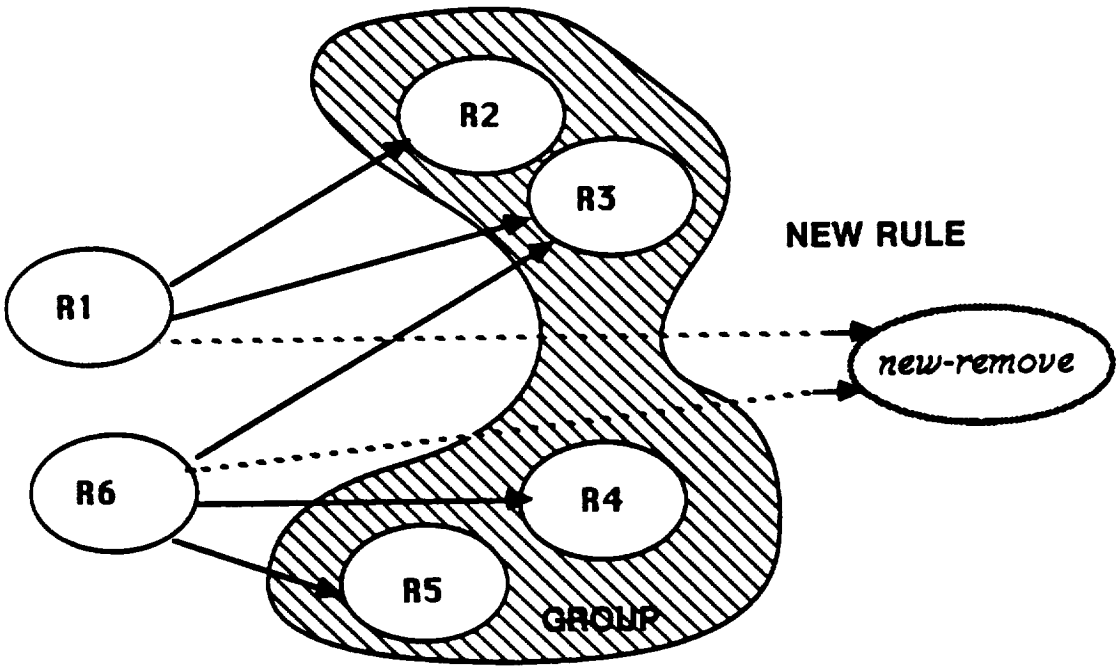

Figure 4.0 *Grouping Rules*

Now, note that it is only possible to remove something from the right hand side of a rule, if the pattern is matched on the left hand side of that same rule. Therefore, the pattern of the element removed in *new-remove* will match some condition in R2 and R3. If *new-rule* fires, and instantiations corresponding to rules R2 and R3 are in the conflict set, the

instantiations will be removed. Thus, the behavior of rules R2 and R3 is altered by the addition of *new-remove*. If R2 and R3 are not fully matched, they will exist in the Rete network. In this case, the *new-remove* will push these rules farther away from the conflict set. Note also that the behavior of R1 is not affected by the addition of *new-remove*.

We can make a similar argument for rules R4 and R5 based on the coupling between R6 and *new-remove*. Simply put, the group of rules affected by the addition of *new-remove* will be those rules with the same parent as *new rule* (i.e. the siblings of *new-remove*).

We therefore see that the group of rules that the new rule could **possibly** affect **directly** is $\{ R_2 , R_3, R_4, R_5 \}$. Naturally the instantiations actually affected by the firing of a rule depends on the data on which a system is operating and the conflict set resolution strategy.

The case for adding a new make rule is similar.The group of rules whose behavior might be affected directly is the set of all immediate successors to the new rule. A new modify, since it is functionally the same as a make and remove, will affect the union of the two groups directly.

Although we have not yet implemented the tool for construction and transversal of the network, its implementation in Lisp should be fairly straightforward. The formal presentation is written mostly in terms of predicates and functions. A dialect of Lisp that supports object-oriented programming could be used to represent nodes by making each node in the net an instance of a "rule class". Methods could be attached to these classes that would retrieve a rule's predecessors, successors, etc. Having devised this general framework for determining coupling between rules, our work is now directed toward implementing this tool and exercising it on the Transformation system.

6 Conclusions

The Transformation system is one of a small group of rulebased systems that has entered into an operational phase. Because the original developer is no longer involved with the day to day operations of the system, the software engineering attributes of the system have become more important. This paper has focused on the nature of rule coupling in the system and has presented a method for understanding the coupling properties of the OPS5 rulebase. We have constructed a general network depicting how rules are coupled within a rulebase, and this network is the basis for deriving a measure of the degree of common coupling for the rules within the rulebase. Furthermore, we have shown how a tool which operates on the principles of this network will allow a maintainer to modify a rulebase with a clearer understanding of how the modification will impact the existing rulebase.

References

[1] Brownston, L., Farrell, R., Kant, E., and Martin, N., *Programming Expert Systems in OPS5*, Addison-Wesley, 1985, pp. 62-64.

[2] Fairley, Richard E., *Software Engineering Concepts*, McGraw-Hill, 1985, pp. 148-151.

[3] Froscher, J. and Jacob, R., *Designing Expert Systems For Ease of Change*, IEE Proceedings of the Expert Systems in Government Symposium, October 1985, pp. 246-251.

[4] Myers, Glenford J., *Reliable Software Through Composite Design*, Van Nostrand Reinhold Company, 1975, pp. 37-39.

[5] Partridge, D. *Engineering Artificial Intelligence Software*, Artificial Intelligence Review, Vol 1., No.1, 1986, pp. 27-41.

[6] Rosenthal, D., *Adding Meta Rules to OPS5 A Proposed Extension*, ACM SIGPLAN Notices, vol 20, no. 10, October 1985, pp. 79-86.

[7] Rosenthal, D., Monger P., Miller G., and Johnston, M., *An Expert System for Ground Support of the Hubble Space Telescope*, Proceedings of 1986 Conference on Artificial Intelligence Applications,NASA Goddard Space Flight Center, May 15, 1986, pp. 43-54.