A Knowledge-Based Selection Mechanism for Control with Application in Design, Assembly and Planning

William F. Punch III[1], Ashok K. Goel[2] and David C. Brown[3]

[1] Knowledge-Based Systems Lab, Computer Science Department

Michigan State University, East Lansing, MI 48824

punch@cps.msu.edu, (517)-353-3541

[2] College of Computing, Georgia Institute of Technology

Atlanta, GA 30332-0280

goel@cc.gatech.edu, (404)-853-9371

[3] Artificial Intelligence Research Group, Computer Science Department

Worcester Polytechnic Institute, MA 01609

dcb@cs.wpi.edu, (508)-831-5618

Abstract. This paper describes a generic, knowledge-based mechanism used to make selections between a fixed set of alternatives. The mechanism, termed *sponsor-selector* has been used as a control mechanism in a number of different knowledge-based systems including design, planning and problem-solver integration applications. We will discuss the general applicability of the sponsor-selector mechanism and describe a number of systems in which has been successfully used.

1 Introduction

Selection is a recurring problem that has occurred throughout work in knowledge-based systems. Knowledge-based selection has two components. The first is the problem of selection itself, that is the design of a mechanism for selecting an item from a fixed set of alternatives. Many early problem-solving systems encountered the selection problem and devised mechanism for dealing with it:

- conflict resolution in production systems like OPS5 [9, 10], which is the procedure for selecting which of several applicable rules to apply next.
- knowledge-source activation as found in many blackboard systems such as BB1 [17].
- priority schedule selection as found in systems like AM/EURISKO [21, 22].

The second problem is the need to encode *meta knowledge* within the selection mechanism that controls *which* selection should be made in a given circumstance.

A number of other researchers have realized the generality of this problem and have addressed it in their systems. For example the SOAR [30] system provides a special condition, the *no preference* condition, which occurs when knowledge is not available for deciding which production to activate next. This condition can invoke a complete problem-solving cycle in a new problem-space to resolve the conflict. Likewise the BB1/BB* [17] system provides a completely separate blackboard, complete with its own set of knowledge-sources, which is used to resolve conflicts concerning which domain knowledge source to activate next.

Of course, other researchers have used selection to solve particular domain dependent problems, such as the selection of materials, components, equipment, people or processes. For example, Bzymek & Della Vecchia [5] describe a system to help customers select plugs for molded case circuit breakers; the system built by Nielson et al [26] selects plastic resins when given information about the application in which they are to be used; Ishii et al [20] describe a methodology and system for process selection in the early stages of product design; and the SafeQual system [23] assists in selecting contractors for a construction project on the basis of their past safety performance and current safety practices.

This paper will describe a knowledge-based selection mechanism called the *sponsor-selector* mechanism. The sponsor-selector mechanism provides knowledge structures within which the selection process can be couched. This mechanism has been used as a part of a number of different tools and applications including design, planning, and control of multiple problem-solvers.

2 The Sponsor-Selector Mechanism

The sponsor-selector mechanism was first used in the routine design tool Design Specialists and Plans Language (DSPL) [2] . The typical structure of a sponsor-selector is displayed in Figure 1. It consists of a hierarchy of three parts: a *selector*, some number of *sponsors*, and associated with each sponsor a *selection item.* Associated with each selection item is a sponsor that provides a measure of how appropriate that item is for selection under the current conditions. Associated with each group of sponsors is a single selector which can be used to resolve ambiguities when multiple items appear equally appropriate and provides a global point-of-view on the selection process.

At any choice point (i.e., some point in the flow of problem-solving at which another item could be selected) the overall process is to run all the sponsors and gather their appropriateness measures for their associated selection items, and then have the selector choose the next item based on the sponsor values and possibly other data. What can be designated as a selection item is not restricted by the architecture, and in fact many strategies have been encoded in this mechanism by deciding

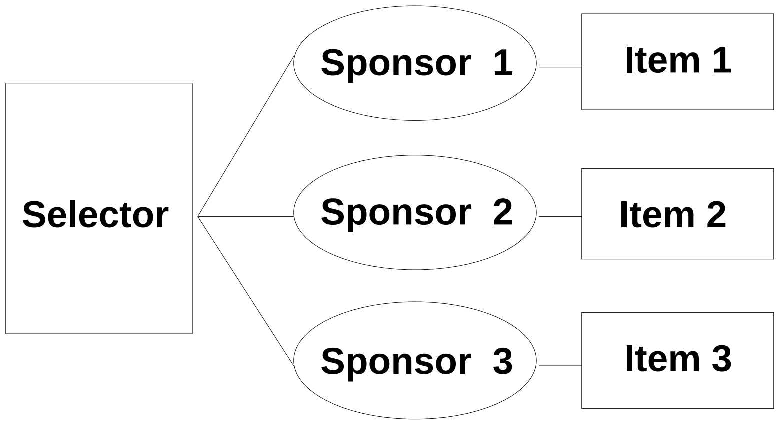

Figure 1: The structure of a sponsor-selector system.

what needs to be selected (i.e., plans, hypotheses, problem-solvers etc.).

The sponsors are thus used as "local" measures of how appropriate their associated items are for achieving the current goal, while the selector takes a more "global" view of selecting which of the applicable items is the most appropriate under the given problem-solving situation.

2.1 Sponsors

Each sponsor contains information about when its associated item is appropriate. The sponsor evaluates this information and yields a discrete value indicating how appropriate the method is. The common form for the sponsor's knowledge is a pattern match group similar to the knowledge groups of Conceptual Structures and Representation Languages (CSRL) [4], a tool used to build hierarchical classification systems, and expanded upon in subsequent papers on structured matching [3] [1]. This representation is a table in which each column, except the last one, is a query about the problem-solving state of the system. Each row is a combination of answers to each question, i.e., a pattern of response. If a row "matches", that is, each pattern element of the row is true for the query with which it is associated, then the last element of the row is the symbolic value for that

[1]Note that we are not restricted to this form of knowledge, in fact other systems have employed different representations.

Resale Value	Cost	Material Strength	Appropriateness Value
High	Medium	Medium	Very Appropriate
Medium	Low	Medium	Slightly Appropriate
?	?	?	Unknown

Figure 2: An example match group the `Wood` sponsor in furniture material selection

pattern match group. Thus the last column is a list of the symbolic values that could be potentially returned by the pattern match group. Control of row examination is of the simple "first true" type; that is, the rows are evaluated in order until one of them matches. Thus the representation allows a knowledge engineer to investigate knowledge patterns in a specific order. Possible patterns of response are ruled out as the rows are evaluated until a matching pattern is found. If no patterns match and all have been examined, then some default value is returned, usually "unknown".

Consider a simple example in the domain of furniture design. Before the design can begin, a material for the piece of furniture must be selected. In this example, there are three possibilities for that material: `Plastic`, `Metal` or `Wood`. There are three criteria on which the selection is based: `Cost`, `Material Strength` and `Resale Value`.

A match group for the `Wood` sponsor is shown in Figure 2. The three criteria are represented as columns in the table, with the fourth column representing an appropriateness rating. Each row represents a pattern of response to the three criteria, along with an appropriateness rating for that pattern of response. For example, the first row represents the pattern of response:

> If it is required that the `Resale Value` be `High` and the `Material Strength` be `Medium` and the `Cost` be `Medium`, then this (Wood) is a `Very Appropriate` selection.

The returned appropriateness values can be any discrete range of symbolic values to represent the appropriateness scale[2]. While this example uses only constants as the matched entities, other cases might require more complicated match expressions.

2.2 Selectors

Each group of sponsors and their associated items are organized under a selector (see Figure 1). The selector does two things:

a. It organizes the set of items that are available to be selected on each cycle of evaluation.

b. It does the selection based on the appropriateness ratings of the sponsors and other knowledge.

When a selector is activated, only the items under it are available for selection each cycle. Note that a sponsor-selector systems can be called recursively by making a sponsored item the invocation of another sponsor-selector. Thus when a sponsor is chosen to invoke its item, that item may invoke yet another sponsor-selector (as well as any other item) allowing a decomposition of knowledge-based choices. Note however that only one sponsor-selector system is "active" (i.e., trying to select an item) at any one time.

A selector is responsible for choosing which of its items to invoke. Its main criterion for so choosing is the appropriateness measures returned from its sponsors, but often this is not enough. The sponsor pattern match information *should* be coded with only local considerations in mind. In other words, encoding the knowledge contained in the sponsor shouldn't require taking account of the other sponsors in order to ensure that only one item is sponsored during a cycle. Ideally, each sponsor is coded with little consideration of other sponsors with more global considerations of proper selection left to the selector.

[2]In this case it is one of 5 discrete values from `Very Appropriate` to `Very Inappropriate`.

Knowledge	Furniture Example
Associated Sponsors	(Wood Sponsor Metal Sponsor Plastic Sponsor)
Ordered Priority List	(Wood >> Metal >> Plastic)
Pattern Match Group	

Style	Selection
Country	Wood
Modern	Plastic

Figure 3: The internals of a selector

There are essentially three categories of decision-making knowledge available to a selector to decide which item to select (Figure 3):

- **Associated Sponsors:** The selection can occur simply based on appropriateness measures. If a *clear* selection is available, i.e., there is exactly one highly rated item, then that item is invoked. If no clear selection is available and no other selection knowledge is available, then a random choice from among the best candidates is selected, i.e., those with the highest appropriateness rating.

- **Ordered Priority List:** If there is a tie among the highest rated items, tie breaking knowledge can be used to choose. That is, the knowledge engineer can provide knowledge of priority which determines which item is selected when appropriateness ratings are not enough. If more than one item is appropriate, then the most important item in the tie group will be chosen. While implementable in a number of ways, one can view this as selecting from an ordered priority list. As shown in a priority list from the furniture example in Figure 3, if two or more materials have the same appropriateness, then we would choose `Wood` first, followed by `Metal`.

- **Pattern Match Group:** For special situations, it is possible to override the normal choice mechanism. This would be similar to the concept of *pathognomonic knowledge* in medicine, that is some knowledge that resolves the choice directly. One implementation would be to use pattern match. If the pattern is matched, then the choice indicated in the matching row is used. If no match occurs, then the default process is used. The advantage here is that previous selection processing can be stored and used directly, saving the processing time of determining the selection during problem-solving. The disadvantage is its inflexibility and narrow applicability. Using the pattern-match group of the furniture selector in Figure 3, if the `Style` is to be `Country` then choose `Wood` over all other materials, it the `Style` is `Modern`, choose `Plastic`.

A sponsor-selector system consists of:

- Local decisions about appropriateness of an item in terms of the present problem-solving state which are coded in the sponsor.
- Global decisions that concern actual choice of an item (based on sponsor results and other information) which are coded in the selector.

As stated previously, multiple groups of sponsor-selectors can and often do exist, but only one such system is active at a time. Others may be chosen by a selector, but only one selector is active at a time.

Completion of cycling through sponsor-selector systems can be accomplished by a *Return* or *Fail* sponsor. These sponsors do not have any associated items. Rather, they indicate when that particular sponsor-selector has finished its work or when it has failed to accomplish the task for which it was activated. These two sponsors are often at the highest priority since failure or completion are the two exit conditions.

3 Comparing Selection and Matching as Tasks

Above, we used the language of structured matching (SM) as part or our description of the sponsor-selection mechanism. However, these two mechanisms are different, having different I/O behavior and different uses.

The goal of matching is to compare two things (A and B) and report on whether they match. Possible answers are Yes, No and Partially – or simple lexical variations on these. In some cases, additional answers may be appropriate. For example, it may also provide, in the "Partially" case, the similarities and differences.

Structured matching is a form of matching, where the process can be subdivided into smaller matching problems. The results from the subproblems are used to determine the overall match. The overall answer to SM is the same as for matching, i.e., Yes, No or Partially.

Structured matching [11, 3] can be used in many situations where the problem is to evaluate a "degree of match" between a data description and the concept the matcher encodes. It has been used in a number of knowledge-based approaches, the first being the hierarchical classifier CSRL [4].

Matching has been used in the sponsor-selector mechanism as part of the sponsor's processing. That is, a sponsor is asked to evaluate the degree of match between the data and the description encoded in the matching knowledge. That description represents the situations in which the object being sponsored can be used. The degree of match therefore represents the appropriateness of its sponsored item given the data. The data is usually a description of the current state of the solution (e.g., a partial design), plus the description of the problem being solved (e.g., the design requirements).

The goal of a sponsor-selector is to take a list of items, or {sponsor, item} pairs, and return the name of the item which appears to be best suited for the current situation. This may involve matching, as certain aspects of the problem situation may need to be detected by matching. However, the task is to select one given many – items in, item out. This isn't the same as matching task, as both the goal and the I/O behavior are quite different. Matchers take data in and yield a degree of match.

The sponsor-selector mechanism, sponsors plus a selector, take data in and yield the most appropriate item for the present data description, not a degree of match. It composes the I/O behavior of its ingredients. Furthermore, the task decompositions are different. In the matching case, one can break matching down into homogeneous subparts, namely sub-matchers, while in the sponsor-selector case, it breaks down into non-homogeneous parts.

Finally, while matching is a natural way to do sponsor evaluation, by matching attributes of the items against the current situation, it is not necessary to do it this way. For example, sponsors for plans in a design situation (see Section 4) could evaluate the appropriateness of an associated plan by doing a rough design or other analysis and by simulating the plan to check appropriateness.

4 Routine Design and the DSPL Tool

As mentioned previously, the sponsor-selection mechanism was first developed in the context of DSPL (Design Specialists and Planning Language) a routine design system [2]. Briefly, the DSPL system addressed the problem of *routine design* [1]. Brown and Chandrasekaran envisioned three general categories of design that a design knowledge-based system might address, which vary depending on whether the knowledge needed and/or the problem-solving to be used are known in advance.

In the simplest class, “routine design”, effective problem decompositions are already available, plans are available for all parts of the design and failure actions are pre-stored. This implies that all the attributes of the design (to be given values as a result of problem-solving) are known at the start of the design activity.

At first glance, routine design might appear trivial, but in fact it is still a knowledge-based activity. Resolving appropriate design decisions with design specifications, available parts, costs etc. requires deliberation, despite the fact that a large part of the design process itself can be described. In fact, it is probably the most common form of design that human designers engage in over their careers.

The structure of a DSPL problem-solver is fairly elaborate, containing a number of levels of abstractions and problem-solving methods associated with each level. However, at the top level a DSPL problem-solver consists of a set of hierarchically organized *design specialists.* A specialist is a design agent that will attempt to design a portion of the overall component. It is a repository for knowledge required for designing that one portion, which it does by utilizing either local knowledge or the cooperative efforts of other sub-specialists. This local knowledge is mainly encoded as a set of *plans* from which the design specialist must select, where a plan is encoded as a sequence of design steps or sub-specialists.

Since we are dealing with only routine design tasks, any plan is a pre-tested sequence that is appropriate for solving a particular kind of problem. Thus the specialist has available stored knowledge of the kind of problem-solving for which each plan is most appropriate and a methodology for selecting the most appropriate plan given the existing problem-solving state. It is in

```
(SPONSOR
    (NAME ExampleDPSponsor)
    (USED-BY ExampleSelector)
    (PLAN ExampleDP3)
    (BODY
       "COMMENT rule out plan if already tried"
       REPLY (IF (ALREADY-TRIED? PLAN) THEN RULE-OUT)
       COMMENT "use qualities to get suitability"
       Qualities
            (TABLE (DEPENDING-ON
                    (RELIABILITY-REQS) (COST_REQS))
               (MATCH
                 (IF (Reliable Cheap) THEN PERFECT)
                 (IF (Medium ?) THEN SUITABLE))
                (OTHERWISE RULE-OUT))
       COMMENT "was Task2 the last failure?"
       Agent (EQUAL 'Task2 LAST-FAILING-ITEM)
       COMMENT "based on the Agent and Qualities variables, rate the plan"
       REPLY
            (TABLE (DEPENDING-ON
                    Agent Qualities)
               (MATCH
                  (IF (T ?) THEN RULE-OUT)
                  (IF (? PERFECT) THEN PERFECT)
                  (IF (? SUITABLE) THEN SUITABLE))
                 (OTHERWISE DONT-KNOW))
    )
 )
```

Figure 4: An example Sponsor in DSPL

the knowledge-based choice of plans for routine design that the sponsor-selector system was first used. In terms of the previous descriptions of the sponsor-selector system, the *items* to be selected are plans. A plan's sponsor yields *appropriateness ratings* in terms of the plan's suitability, user requirements and the problem-solving state. The selector chooses which plan is most appropriate, if any, or signals failure if no plan is appropriate or if all plans have already been tried and none has succeeded.

4.1 Sponsors

A typical DSPL Sponsor is shown in Figure 4. A DSPL Sponsor yielded appropriateness rating in a fixed scale of *(perfect, suitable, don't know, not suitable, ruled out)*. In DSPL, Sponsor knowl-

```
(SELECTOR
    (NAME ExampleDPSelector)
    (USED-BY Example)
    (TYPE Design)
    (USED DPSponsor1 DPSponsor2)
    (BODY
        COMMENT "if Plan 26 is perfect, use it"
        REPLY (IF (MEMBER 'Plan26 PERFECT-PLANS)
               THEN 'Plan26)
        COMMENT "if there are perfect plans, use in preferred order"
        REPLY (IF PERFECT-PLANS
               THEN (DESIGNER-PREFERENCE
                     PERFECT-PLANS)
               ELSE NO-PLANS-APPLICABLE)
     )
)
```

Figure 5: An example Selector in DSPL

edge comes in two forms. The first is "immediate" knowledge which is essentially single rules that evaluate the appropriateness of the plan. For example, the first `REPLY` form represents "If the plan has already been tried, then rate it's appropriateness as 'ruled-out'. The second form of knowledge used several queries to establish the appropriateness of a plan. The "accumulated" knowledge was represented as a pattern-match table as previously described. In Figure 4, the first table represents:

> If the `RELIABILITY-REQS` (reliability requirements) are `Reliable` and the `COST-REQS` (cost requirements) are `Cheap`, then this plan is `PERFECT`, otherwise if the reliability requirements are `Medium`, then regardless of the value of cost requirements (the `?`) the plan is `SUITABLE`, otherwise the plan is a `RULE-OUT`.

In fact the only real difference between the two kinds of knowledge appear to be the fact that the immediate knowledge/rules examined problem-solving history values whereas the accumulated knowledge represented actual plan suitability.

4.2 Selectors

A typical DSPL Selector is shown in Figure 5. This figure shows two of the three kinds of choice categories described in Section 2.2. The first `REPLY` form is an example of pathognomonic knowledge: "regardless of the normal forms of selectivity, if `Plan26` is perfect then choose it". The second kind of knowledge is an example of tie-breaking knowledge between multiple perfect plans: "If there are `PERFECT-PLANS`, then use the plans according to `DESIGNER-PREFERENCE`" (i.e., use the order of plans given by the designer as it was encoded into DSPL). As for representing the first kind of selector knowledge, selection based on only sponsor-results, that default behavior must *also* be encoded by the knowledge engineer.

4.3 Practical Applications of Routine Design

The sponsor-selector approach is an integral part of the DSPL language. DSPL has been used to successfully develop knowledge-based systems in a variety of real-world areas.

The AIR-CYL system [2] designs an Air Cylinder given specific requirements. This component was selected from a real industrial situation, and was intended for the accurate and reliable movement of a shutter across a beam of radiation. The routine design problem was decomposed into subproblems at various levels of detail, each represented by a Specialist in the DSPL language. For example, the "AirCylinder" problem had "Spring", "Head" and "The Rest" subproblems. One or two plans were associated with each Specialist. This particular problem did not require additional plans. In addition, the plans were used in a preferred order for each subproblem. The sponsor-selector mechanism was able to support this requirement quite easily, and the AIR-CYL system was able to produce acceptable designs.

The EDES system [25] is capable of designing multi-stage Operational Amplifiers. Given specifications, such as input impedance and gain, the system produces a design for a working Op-Amp. The circuits designed by EDES were reviewed by human experts, and the results were verified using the SPICE simulator. As with all DSPL-based systems, the problem is decomposed into subproblems by selecting plans. Here the top-level decomposition is into "stages". The subproblems have multiple plans, representing the different circuits used to implement the stages. Plan selection is accomplished by giving preference to the plans representing the simpler circuits. Thus the sponsor-selector mechanism will use the simplest circuit for each stage if at all possible.

Another practical DSPL-based system is GDES [33]. The GDES expert system designs Gear Pairs, a very frequently used component of mechanical systems. The size and shape of gears change with the power and performance requirements. Gears need to be designed for three goals — strength, dynamic load and wear. In this problem the decomposition was based on expert heuristic knowledge about the order in which these goals should be addressed. Previously it had been suggested [24] that this problem was best solved by Iterative Refinement of a complete (although initially incorrect) design. However, heuristic decomposition, as implemented by the sponsor-selector mechanism, allows DSPL to be used to solve the problem. Plans are also used in this system to respond to variations in requirements. For example, material may or may not be specified by the user. This results in a different approach to the problem. The knowledge-based sponsor-selector approach allows this situation to be detected and the correct plan to be preferred.

5 Abductive Assembly and the PEIRCE Tool

5.1 The RED system

Another application of the sponsor-selection mechanism occurs in the RED [31, 18, 19] system which provides a test-bed for experiments with the methodology of *abductive assembly* [3].

[3]The book titled "Abductive Inference: Computation, Philosophy, Technology" [19] traces the history of this development in detail.

RED identifies blood-antibodies present in patient sera so that a non-threatening transfusion blood can be given to that patient. RED's task is to determine the antibody make-up of a patient's serum based on the results of tests on that serum against red blood cells (RBCs) with known antigens. In short, RBCs with known antigens are placed in samples of the serum to be tested. Those RBCs that "clump" are reactive, that is the patient has antibodies to one or more of the antigens on those cells. Analysis of the results of these tests is an abductive problem, namely explaining the reactivity of the serum (as measured by clumping with known RBCs) in terms of hypotheses about the the patient's circulating antibodies.

Essentially, the RED systems is a two stage process. The first stage evaluates the plausibility of the antibodies that could be present in the patient serum. This was accomplished using the technique of *hierarchical classification* [4]. The hierarchy used represents a taxonomy of antibody types, with specific antibodies at the leaf nodes. Associated with each antibody node in the hierarchy is pattern match information that can roughly determine the plausibility of the antibody based on patient reactivity information and knowledge about the antibody itself.

The hierarchical classifier examines the hierarchy of antibodies in a top-down fashion. If an antibody *establishes*, that is, the antibody's pattern match knowledge determines that its plausibility is above some threshold of acceptability, then all the antibodies directly below the established one are also examined i.e., the parent node is *refined.* The result obtained from the hierarchical classifier is a list of plausible antibodies and their associated plausibilities.

This list is then used by the second stage of the RED process, the assembly stage. In assembly, we take a set of *findings* that need to be *explained* and a set of hypotheses that we can use to explain those findings, and select from the hypothesis set some subset, the *compound explanation*, that explains (covers) the findings and meets other conditions (parsimony of the compound explanation, compatibility of the compound explanation members, maximal plausibility of the compound explanation members etc.). In the RED case, the findings are the clumping reactions (or lack of reaction) of the serum with particular RBCs. The hypotheses used to explain these findings concern the existence/absence of antibodies in the patient serum. The set used is the list delivered from the hierarchical classifier. The explanation process then is to assemble a compound explanation (a subset of antibody hypotheses) that explains/covers the observed findings plus meets other user-defined requirements.

The algorithm used for assembly is roughly as follows:

1. Select a single finding that needs to be explained.
2. From the antibody hypotheses found by the hierarchical classifier, select those hypotheses that offer to explain that finding.
3. If only one hypothesis offers to explain the finding, select that hypothesis, otherwise select the most plausible hypothesis of the set. Again, plausibility is pre-determined by the hierarchical classifier.
4. Integrate the selected hypothesis into the compound explanation. If the introduced hypothesis is *incompatible* with the existing compound, then there are two choices. Either select another hypothesis (i.e., go back to step 3) or remove the incompatible ones in the compound explanation. Maintain a history mechanism to avoid loops. Note that if a hypothesis is removed, a finding might need to be explained again, perhaps by a different hypothesis.
5. Update any other findings that might be explained by the integration, and if some findings still remain unexplained, go to 1. Else quit.

5.2 Problems of Control

Our experiments with the RED system identified a number of problems with the approach. These essentially were the result of our inability to fully control the algorithm so that various experiments on the abductive assembly approach could continue. For example, one experiment we wished to conduct would form compound explanations from each level of hypotheses in the hierarchical classifier. Thus one could get an explanation at a coarse level of detail initially then refine that explanation with more refined hypotheses from the classifier. However, it required enormous work to reprogram the system each time a change was desired. The solution to this problem was to analyze the entire algorithm and identify its functional parts or *methods*. Furthermore, we needed to provide a control approach that would allow both easy addition of new methods and easy modification of when a method should be used so that we could more easily experiment with new strategies of abduction.

The result of this need was the PEIRCE system [29]. The basis for the control decisions in PEIRCE, i.e., which abductive assembly method to run next, was made using the sponsor-selector system. Thus instead of selecting plans in a design system, PEIRCE used the sponsor-selector as a control mechanism for determining the appropriateness of methods for the problem-solving context. The abductive assembly algorithm was therefore broken down into the goals shown in Figure 6.

5.3 Sponsors and Selectors in PEIRCE

The encoding of knowledge about the appropriateness of methods used a pattern-match table as described in section 2.1. Two kinds of knowledge were used by the sponsors to capture method appropriateness:

1. Information about what methods have been applied so far in the problem solving process, such as which methods have executed so far, if at all, and when they executed. The PEIRCE system

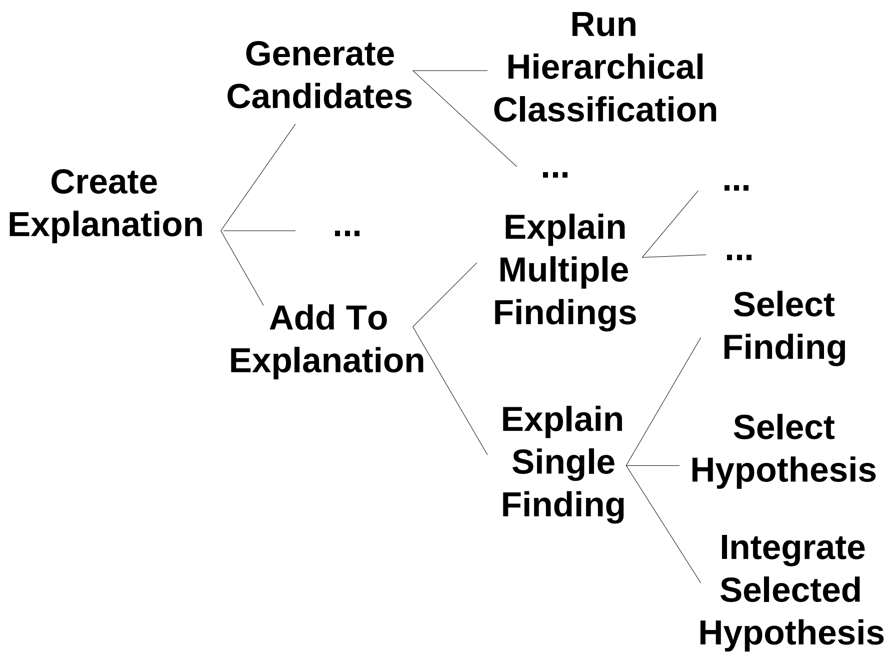

Figure 6: A goal structure for abduction

had an history recording mechanism that provided access to these kinds of information.

2. Information concerning goal achievement, such as, "Is the explanation complete?" or, "Has this finding been explained?". This kind of information has to be supplied by the programmer who defines both the methods and the pattern match knowledge. It is not part of PEIRCE.

Selectors encoded all three kinds of knowledge described in section 2.2. The default behavior is to select a method based on its sponsor value, the method with the highest appropriateness is the method next executed in the abduction problem-solving process. We also modified this approach slightly by establishing a minimum appropriateness difference, termed the *clear winner* strategy. If the difference in appropriateness values between the most appropriate and second-most appropriate method was less than this difference, then the methods were still considered to be "tied". If a "tie" did occur and a tie break was needed, each selector contained an *ordered priority list* that was used to break the tie. Essentially, the method that is invoked next is the method that is one of the "tied" methods and shows up earliest in the priority list. Finally, the user could optionally provide pathognomonic knowledge about what method to invoke next. This knowledge was in the form of a pattern match table as discussed in Section 2.2. If such knowledge was provided, the table was evaluated *first* to determine if the conditions it encodes exist. If they do, then the normal selection procedure described above is ignored and the results of the pattern-match table are used to make the selection.

5.4 Evaluation of Peirce

Our first evaluation of the Peirce system was done by re-implementing the original Red system. Red was re-coded as a set of methods with associated sponsors that would allow changes of strategy of operation of the abductive assembler. In particular, we used as a point of comparison a set of 20 cases [31] that we had run previously in Red, and whose results were confirmed by a panel of experts . Three specific variations on the Red system were proposed for examination:

1. Providing some interaction between the hierarchical classifier and the abductive assembler for refinement control (see Section 5.2).

2. Postponing "hard decisions", namely postponing which hypothesis to integrate into the compound explanation when the choice was not simple, for as long as possible. This was a "least commitment" approach to abductive assembly.

3. Providing for abductors that specialize in explaining particular kinds of domain findings. This would allow for divide-and-conquer strategies by allowing sub-abductors to explain different parts of the findings in the problem at hand.

Strategies were changed for each of the experiments by changing the appropriateness knowledge in the sponsors of the system. We found it fairly simple to change *when* a method is invoked in the overall assembly algorithm by modifying the knowledge of when it is appropriate. Such changes are local since they affect only the invocation of one method but have a global affect on the assembly algorithm. Adding new methods is equally simple since one need only include it in the sponsor-selector with some knowledge of when it is appropriate.

6 The TIPS Architecture for Problem-Solver Integration

The experience with Peirce led to investigations on how broadly applicable a sponsor-selector system might be as a control structure.

The motivation for this work comes from experience with the *generic task* (GT) [6, 7] approach to system building. Briefly, the GT view emphasizes the identification of domain-independent, generic approaches to solving particular kinds of high-level problems (e.g., classification, routine design). Once a task is identified, the problem is to identify forms of knowledge for both representation and control that are appropriate for that particular task. The promise of such an approach is that once a robust set of tasks have been identified, along with the tools that encode the knowledge for that task, they could form the building blocks of more complex problem solving. That is, complex problem solving could emerge from the proper integration of generic task problem-solvers.

In the first generation of work on using GTs to build diagnostic and other complex problem solving systems (such as MDX2 [32] and Red) the invocation of a GT problem solver was pre-programmed as part of the task-structure. In other words, the invocation of GT problem-solvers had been implicitly “hard-wired” during the programming of the system. It was later realized that in order to use multiple methods to achieve problem-solving goals we needed method selection to occur at run-time based on the state of the problem and availability of knowledge.

Therefore the goal was to design a general-purpose architecture that allowed integration of high-level problem-solvers based on a dynamic problem state. The result was a methodology to analyze complex problems using the *task structure*, an identification of both the goals in the complex problem and the capabilities to meet those goals in each sub problem-solver. The TIPS problem-solver is an architecture that directly captured that analysis.

The TIPS approach to creating dynamically integrated problem-solvers is not a full problem-solving system in itself. It provides only enough mechanism to monitor goal achievement and to select methods to achieve active goal. The design of these methods is left to the knowledge engineer. TIPS only requires that each method "inform" the system as to what it does during execution in terms of "goal achievement". This allows a knowledge engineer to take advantage of "tried and true" software (such as existing generic task problem-solvers) without converting it to another format and allows a diversity of methods and representations for different kinds of problem-solving.

6.1 Sponsors and Selectors in TIPS

Once again, TIPS used the sponsor-selector system as the control mechanism. The approach was exactly as in PEIRCE except that the selected items were much larger problem-solving entities. For example, in TIPS selection is between problem-solving entities like Hierarchical Classification or Abductive Assembly.

6.2 Application of TIPS to Diagnosis

TIPS was first used to create a medical diagnosis system called Med-TIPS. Med-TIPS deals with a broad range of liver and blood diseases and their various interactions. The following description emphasizes two aspects of the Med-TIPS application. First, it briefly describes the kinds of problem-solvers that Med-TIPS had available to solve the diagnosis problem. Second, it describes the sponsor knowledge that was used to decide when that problem-solver is appropriate (see [27] for more details).

- **Compiled knowledge diagnostician**. This module is responsible for creating diagnostic explanations using the abductive-assembly/hierarchical-classification architecture found in Red. It is used to meet the Diagnose-using-Compiled-Knowledge goal.

  Briefly, this module has two subgoals, Hierarchical Classification and Abductive Assembly. As in Red and Peirce, the problem-solver solver examines a hierarchy of (in this case) malfunction categories to determine the plausibility of malfunctions given the present circumstances. This plausibility information is used in the next phase of abductive assembly to assemble a subset of the malfunction hypotheses such that the resulting *explanation* is most plausible, consistent and covers the findings.

  Sponsor-Knowledge: The module is appropriate under the following four conditions: as the very first module in a run (to create an initial diagnosis); if more data has been gathered; if a datum value was shown to be invalid and replaced was by a validation method; or if deep/causal reasoning has suggested a particular diagnosis.

- **Data gathering module**. This module gathers evidence that can establish or rule-out pertinent diagnostic hypotheses. It is used to meet the Gather-Data goal.

  In this particular case, the data that is needed next is determined by the nodes in the hierarchical classifier that suspended due to lack of data. If this data can be made available, then further exploration of the hierarchy is enabled.

  Sponsor-Knowledge: The module is appropriate when: the hierarchical classifier cannot explore leaf level hypotheses due to lack of data, or one of the potential hypotheses in a hard decision (see item "Redoing hard decisions" in this section) has an unanswered question in its specialist.

- **Data validation module**. The module checks the validity of certain data that have been questioned during the run of the compiled knowledge diagnostician module. It is used to meet the Validate-Data goal.

  Described in some detail elsewhere (see [8]), the idea is as follows. Rather than relying strictly on statistical "averaging" based on multiple sensor readings for one datum, this module uses expectations derived from partial hypotheses already formed by diagnosis. If these expectations are not met (i.e., a partial conclusion of "Liver Disease" expects some changes to occur in observed liver enzyme values, but those values are presently observed to be normal) then there is a potential data problem. These unmet expectations are flagged and further investigated by specific test procedures that validate their value.

  Sponsor-Knowledge: The module is appropriate if, in the process of diagnosis, a plausible composite explanation is generated whose data expectations are not met. Those data that do not meet expectations are noted as questionable and are submitted for validation.

- **Redoing hard decisions module**. The module examines the assembly of alternate diagnostic explanations stemming from a hard decision (or decisions) in the compiled knowledge diagnostician module. It is used to meet the Change-Hard-Decisions goal.

  A hard decision is a situation reached in abductive assembly when no clear criteria is available to differentiate among a set of hypotheses that offer to explain a finding. For example, the hypotheses A, B and C all offer to explain finding F and all have the same plausibility. If the only criteria available for differentiating which is the "best" hypothesis to explain F is plausibility, then there is no basis for a choice and a hard decision occurs. Typically the system makes a random choice and goes on but it also records the system state at that point so that the user may later go back and explore other possible solution paths.

Sponsor-Knowledge: The module is appropriate when a hard decision has resulted during the abductive assembly process.

- **Causal reasoning module**. The module applies the coherence view of causal modeling of [28] to the diagnostic explanations generated by the compiled knowledge diagnostician module. It is used to meet the Do-Causal-Reasoning goal.

  Briefly, associated with each diagnostic hypothesis is some generic knowledge about how the malfunction it represents modifies normal function. These changes are broadly categorized as functional changes (e.g., Anemia causes loss of oxygenation function), connective changes (e.g., blocked Common Bile duct causes back-up of bile) and structural changes (e.g., swelling of pancreas can push on the liver). If a possible link between the elements of a partial abductive assembly hypothesis are found, their relationship is explored using a model of the involved elements to determine some causal results not yet available to the compiled diagnosis component.

  Consider an example from Med-TIPS. If hemolytic anemia causes a functional increase in iron products in the blood and the liver is responsible for clearing these products, a relationship between hemolytic anemia and the as-of-yet non-specific liver disease can be explored to discern a possible disease process.

  Sponsor-Knowledge: The causal reasoning module is appropriate when a possible causal interaction has been noted between hypotheses of the present explanation.

The Med-TIPS system was evaluated on published pathology reviews of difficult cases [16]. Our main goal was to test the system's ability to discover the correct answers (as published in the reports) using different sequences of problem-solvers based on the kinds of data and knowledge given to Med-TIPS. For example, we would remove some causal-knowledge and observe how the system would try to solve the problem using data-gathering techniques. This system was fairly robust in terms of finding similar answers using different kinds of problem-solvers.

7 Robot Planning and the Router System

Router is a robot path planning system that generalizes the sponsor-selector mechanism and the TIPS architecture to accommodate some issues related to learning. The sponsor-selector mechanism and the TIPS architecture do not explicitly address the issues of acquisition of domain knowledge. Given a task (or goal) structure, specific methods and the knowledge they use, as well as appropriateness measures for selecting a method at run-time, the TIPS architecture results in the selection of the same method each time the same goal is presented to the system. This is because while TIPS selects methods dynamically, the domain knowledge and the appropriateness measures in TIPS are static. Of course, in general, an agent's domain knowledge and appropriateness measures may change over time. The Router system investigates the issue of run-time method selection as domain knowledge grows and changes over time. It uses the sponsor-selector language to represent meta-knowledge for selecting methods.

Router is a robot path planning and learning system. It operates on a representation of physical spaces such as the Georgia Tech campus. The system has evolved in three stages: Router1, Router2, and Router3. Router1 uses a hierarchically-organized topographic model of the physical space to plan routes [14]. It plans routes by heuristically searching the model in a top-down manner. We will call this the model-based method, $M1$. In contrast, Router2 uses past planning experiences (or cases) to plan new routes [13]. It plans routes by retrieving, adapting, and combining routes planned on previous occasions. We will call this the case-based method, $M2$. Also, Router2 chunks newly-planned routes into cases and stores them in its case memory for potential reuse in future.

In this way, Router2's case base grows as it plans new routes.

Router3 integrates Router1 and Router2 [12]. The architecture of Router3 is a generalization of the TIPS architecture. Given the same task twice, TIPS would choose the same method on both. However, Router3 can choose different methods even for instances of the same task depending on the population of cases in its memory. This memory continually grows as the system solves new problems. Given a specific task instance, Router3 can potentially use either the model-based method instantiated in the Router1 system, or the case-based method, instantiated in Router2. So the issue is what criteria to use for selecting between the model-based and case-based methods given that Router case base grows with planning experiences?

Goel and Chandrasekaran [15] describe three general criteria for selecting a method:

- *Properties of the Solution*: Different methods produce different types of solutions. Some methods may produce optimal solutions, while others may produce satisficing ones. Some methods may produce precise answers, while the answers of others may be only qualitative. What, then, are the requirements on the properties of a solution to the given task?

- *Properties of the Process*: Different methods may require different computational resources. Some methods may be computationally so complex that they are pragmatically infeasible. Others may vary in the processing time and memory space they take. What, then, are the constraints on the availability of computational resources?

- *Properties of Knowledge*: Different methods use different types of knowledge. Some methods may use knowledge that is not available in the given problem domain. What types and forms of knowledge, then, are available in the problem domain? How well does the available knowledge match the knowledge used by the various methods applicable to the task?

If a multi-strategy system (or agent) has meta-knowledge of this kind, then the system can use this knowledge for selecting between the methods, even as the system's domain changes and grows.

Given these criteria, the issue becomes how do Router's model-based method $M1$ and case-based method $M2$ compare along these three dimensions. First, the solutions produced by $M2$ can be better or worse than the solutions generated by $M1$, depending on the degree of similarity between the retrieved case and the task, and the mechanism for adapting the case to meet the specification of the task instance. Second, if a similar case is available in the case memory, then, in general, $M2$ is computationally more efficient than $M1$. This is because the stored case immediately provides a solution in the "neighborhood" of the desired solution. Third, $M1$ is more generally applicable than $M2$ because $M2$ requires that a case similar to the present task be available in the case memory.

Router3 uses the sponsor-selector mechanism to encode these selection criteria: it invokes $M2$ if a similar case is available, otherwise it selects $M1$. Also, Router3 chunks the solution generated by either method as a new case and stores it the case memory for use by $M2$ during the next problem-solving cycle.

As a result, Router3's style of reasoning shifts between case-based and model-based. Router3 can start solving a given problem by the case-based method, and adapt a retrieved case using the model-based method. Here, two different methods are used within the context of solving a single task. In addition, Router3's reasoning shifts from one style to another across a sample of task instances. For example, Router3 initially may have few cases in its case memory, and therefore may rely almost entirely on the model-based method. However, it automatically chunks the solutions generated by model-based method into cases and stores them in the case memory for reuse by the case-based method. As the number of cases in the case memory increases, Router3 increasingly favors method $M2$. The shift in Router3's style of reasoning is smooth and graceful, not abrupt or jerky.

8 Conclusions

We have described the sponsor-selector mechanism for solving the selection problem and have reviewed a number of applications of this mechanism in knowledge-based systems. These applications show the utility of the sponsor-selector mechanism in the context of knowledge-based selection from a set of fixed alternatives for routine design, medical problem-solving and routine route planning.

This does have a number of inadequacies, however, in particular from the control point of view. The sponsor-selector is a direct representation of the goals and their relationship in a problem such as diagnosis. In implementing this in, say, the TIPS system, those goals are only represented implicitly in the sponsor-selector system. In other words, there is no *direct* representation of the goals or of when the goals become active, or of when a goal has been achieved, etc. At present, each sponsor must contain knowledge concerning when its goal(s) are appropriate for exploration and under what conditions its method is appropriate for achieving those goals. Moreover, this lack of clean separation leads to confusion as to what gets sponsored. For example, Do-Diagnosis in Med-TIPS is a goal but Hierarchical Classification is more of an approach. This is one of the more pressing problems that needs to be addressed.

Despite these problems, it is clear that the sponsor-selector mechanism is of general utility and may used as both a control structure and a pure selection mechanism.

References

[1] D.C. Brown. Routineness revisited. In *Mechanical Design: Theory and Methodology*. Springer-Verlag, 1993. *to appear.*

[2] D.C. Brown and B. Chandrasekaran. *Design Problem Solving: Knowledge Structures and Control Strategies*. Pitman, London, UK, 1989.

[3] T. Bylander, T. Johnson, and A. Goel. Structured matching: A task-specific technique for making decisions. *Knowledge Acquisition*, 3(1):1–20, 1991.

[4] T. C. Bylander and S. Mittal. CSRL: A language for classificatory problem solving and uncertainty handling. *AI Magazine*, 7, Summer 1986.

[5] Z. M. Bzymek and D. Della Vecchia. Product selector expert system. In G.Rzevski & R.A.Adey D.G.Grierson, editor, *Applications of AI in Engineering, VII*, pages 1131–1139. Elsevier Applied Science, 1992.

[6] B. Chandrasekaran. Towards a functional architecture for intelligence based on generic information processing tasks. In *Proceedings of the International Joint Conference on Artificial Intelligence 87*, pages 1183–1191. International Joint Conference on Artificial Intelligence, 1987.

[7] B. Chandrasekaran. Design problem solving, a task analysis. *AI Magazine*, 11(4):59–71, 1990.

[8] B. Chandrasekaran and W. F. Punch III. Data validation during diagnosis, a step beyond traditional sensor validation. In *AAAI87*, pages 778–782, 1987.

[9] C. Forgy and J. McDermott. OPS: A domain-independent production system language. In *IJCAI*, volume 5, pages 933–939, 1977.

[10] C. L. Forgy. The OPS5 user's manual. Technical Report CMU-CS-81-135, Computer Science Dept. Carnegie-Mellon University, Pittsburgh, Pa, 1981.

[11] A. Goel and T. Bylander. Computational feasibility of structured matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(12):1312–1316, December 1989.

[12] A. Goel and T. Callantine. A control architecture for run-time method selection. In *Proc. AAAI 1991 Workshop on Cooperation Among Heterogeneous Intelligent Systems*, pages 24–28, Anahiem, CA, July 1991.

[13] A. Goel and T. Callantine. An experience-based approach to navigational path planning. In *Proc. IEEE/RSJ International Conference on Robotics and Systems*, pages 705–710, Raleigh, North Carolina, July 1992.

[14] A. Goel, T. Callantine, M. Shankar, and B. Chandrasekaran. Representation, organization, and use of topographic models of physical spaces for route planning. In *Proc. Seventh IEEE Conference on Artificial Intelligence Applications*, pages 308–314, Miami, FLA, February 1991. IEEE Computer Society Press.

[15] Ashok Goel and B. Chandrasekaran. *Artificial Intelligence Approaches to Engineering Design, Volume II: Innovative Design*, chapter Case-Based Design: A Task Analysis. Academic Press, 1992.

[16] A. M. Harvey and J. Bordley III. *Illustrative Case I in Liver Diseases*, pages 294–298. W. B. Saunders, 1972.

[17] B. Hayes-Roth. A blackboard architecture for control. *Artificial Intelligence*, 26:251–321, 1985.

[18] J. R. Josephson, B. Chandrasekaran, J. R. Smith, and M. C. Tanner. A mechanisim for forming composite explanatory hypotheses. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-17(3):445–454, 1987.

[19] John Josephson and Susan Josephson. *Abductive Inference: Computation, Philosophy, Technology.* Cambridge University Press, 1993. forthcoming.

[20] K.Ishii, C.H.Lee, and R.A.Miller. Methods for process selection in design. In J. Rinderle, editor, *Proc. Design Theory & Methodology Conf.*, pages 105–112. ASME, Sept 1990.

[21] D. B. Lenat. AM: An artificial intelligence approach to discovery in mathematics as heuristic search. Technical Report STAN-CS-76-570, Stanford University, 1979. Reprinted in R. Davis and D.B. Lenat. *Knowledge-based system in artificial intelligence*, NewYork: McGraw-Hill, 1980.

[22] D. B. Lenat. EURISKO, a program that learns new heuristics and domain concepts. *Artificial Intelligence*, 21:61–98, 1983.

[23] R. E. Levitt. Howsafe: A microcomputer-based expert system to evaluate the safety of a construction firm. In Celal N. Kostem and Mary Lou Maher, editors, *Proceedings of Expert Systems in Civil Engineering*, pages 55–66, ASCE Spring Convention Seattle, WA, April 17 1986. Technical Council on Computer Practices of the American Society of Civil Engineers.

[24] T. Mitchell and N. Langrana. Progress towards a knowledge based aid for mechanical design. In *Symp. on Integrated and Intelligent Manufacturing.* ASME W.A.M., 1986.

[25] E. Muntasser. *An Electronic Design Expert System.* Master's thesis, Worcester Polytechnic Institute, May 1990.

[26] E. H. Nielsen, J. R. Dixon, and M. K. Simmons. GERES: A knowledge based material selection program for injection molded resins. In *Proc. ASME Conf. on Computers in Engineering, Vol.1*, pages 255–261, 1986.

[27] W. F. Punch III. *A Diagnosis System Using a Task Integrated Problem Solving Architecture (TIPS), Including Causal Reasoning.* PhD thesis, The Ohio State University, 1989.

[28] W. F. Punch III. Interactions of compiled and causal reasoning in diagnosis. *IEEE Expert*, 7(1):28–35, 1992.

[29] W. F. Punch III, M. C. Tanner, J. R. Josephson, and J. W. Smith. Using the tool PEIRCE to represent the goal structure of abductive reasoning. *IEEE Expert*, 5(5):34–44, 1990.

[30] P. S. Rosenbloom, J. E. Laird, and A. Newell. SOAR: an architecture for general intelligence. *Artificial Intelligence*, 33:1–64, 1987.

[31] J. W. Smith, J. R. Svirbely, C. A. Evans, P. Strohm, J. R. Josephson, and M. C. Tanner. Red: A red-cell antibody identification expert module. *Journal of Medical Systems*, 9, Issue 3,:121–138, 1985.

[32] J. Sticklen. *Distributed Abduction.* AG Scientific, Basel, Switzerland, 1989.

[33] U. Trivedi. *An Expert System for Gear Train Design.* Master's thesis, Worcester Polytechnic Institute, September 1988.

List of Figures

1 The structure of a sponsor-selector system. 4

2 An example match group the `Wood` sponsor in furniture material selection 5

3 The internals of a selector 7

4 An example Sponsor in DSPL 12

5 An example Selector in DSPL 13

6 A goal structure for abduction 19

List of Footnotes

Manuscript received ...

1. **Financial Support**: Bill Punch's research has been supported by the Ameritech Foundation and the Center for Microbial Ecology (Michigan State).

   Ashok Goel's work at Georgia Tech has been supported by International Business Machines and Georgia Tech's Office for Interdisciplinary Research.

   Dave Brown's research at Worcester Polytechnic Institute has been supported by Digital Equipment Corporation.

2. Note that we are not restricted to this form of knowledge, in fact other systems have employed different representations.

3. In this case it is one of 5 discrete values from `Very Appropriate` to `Very Inappropriate`.

4. The book titled "Abductive Inference: Computation, Philosophy, Technology" [19] traces the history of this development in detail.