Interactive Explanation for Cooperative Information Systems

Michael J.Minock **Wesley W. Chu**

minock@cs.ucla.edu wwc@cs.ucla.edu

Computer Science Department, University of California at Los Angeles

Cooperative Information Systems let users pose imprecise queries and receive approximate or summary answers in return. Yet cooperative answers should be accompanied by an explanation of how they were derived. We present an explanation system for the cooperative information system CoBase[1][3]. The architecture and formalism for this explanation system are the main focus of this report. In addition this report touches on the more general problem of how to interactively refine and extend explanations.

CoBase's explanation system provides multimedia, interactive, user-sensitive and context-dependent explanations of CoBase cooperative operations. The explanation system requires access to queries, CoBase knowledge structures, execution traces, and answers. The explanation system applies classification rules to interpret this information and generation rules to produce explanations.

The formal model of explanation generation consists of the representation for the subject of explanation and the definition of the classification and generation mechanism. The representation is a type of semantic network, but includes several enhancements for explanation. The classification and generation mechanism is defined by a set of inference rules. This includes a selection predicate which uses a heuristic rule override graph to decide which generation rule to apply based on the user, CoBase context, and generation context.

This work touches on explanation refinement and extension through interaction. Users indicate which patterns of explanation they dislike or prefer. These indications are reflected in the heuristic rule override graph, enabling the system to produce preferred explanations in the future. Users may also extend explanations. Through a series of editing operation (change, delete, expand, or move) users transform explanations. From these transformed explanations new generation rules are induced. These induced rules are added to the generation rulebase.

1.0 Introduction

Systems must become better at explaining the operations they offer, the actions they take, and the results they yield. Currently many systems do this to one degree or another. Word processors and spreadsheets offer simple, self-explanatory operations and show direct results immediately. Information systems offer custom query forms and data-entry screens and display simple answer set presentations or reports. These systems have succeeded in their communication responsibilities by simplifying the set of operations they offer and by presenting the application state graphically in accordance with familiar, intuitive metaphors. Yet in many instances we wish to grant systems broader, more complex responsibilities. Simple communication techniques are insufficient in these cases. If required, such systems must be able to explain themselves.

One area in which explanation technology is required is in Cooperative Information Systems. Traditionally database systems accept precise query specification (SQL, Datalog, etc.) and return exact answer sets. In Cooperative Information Systems people pose imprecise queries and receive approximations when answer sets are non-existent and summaries when answer sets are large. Yet these cooperative answers should be accompanied by an explanation of how they were derived. CoBase [1][3] is a cooperative information system, providing approximate, associated and summary answers to a user's relational and object-oriented queries. CoBase uses the Type Abstraction Hierarchy (TAH)[2] as its principle knowledge structure to guide the query transformation process. This work addresses the use of explanation technology in the CoBase system.

1.1 Background

1.1.1 Cooperative Information Systems

There have been numerous Cooperative Information Systems proposed in the research community. A survey of such systems is presented in [5]. Initial attempts relied on domain specific knowledge encoded in complex rules. As the field has progressed, knowledge representations have been simplified and knowledge acquisition techniques have been bought to bear [2][10]. Domain independent systems are now feasible. Still there has been very little done on generating explanation for Cooperative Information Systems.

1.1.2 Explanation

The most common and rudimentary form of explanation is the placement of print commands within a source code or rulebase. This method may be extended by forming a library of canned templates that are conditionally printed based on flag variables. The flag variables may depend on a variety of factors including user skill and interest. Yet such explanations mirror the flow of control of the program and are only useful for people debugging the system, or users with detailed understanding, tolerant of much redundant and irrelevant description. The fundamental problem with such approaches are that they do not have access to all the information required to produce an explanation. A print command has access only to local and global variables. Because of this dependence on the program variable stack, flagged attempts are only capable of explaining things as they happen.

To decouple system action and explanation, it is necessary to form a trace of the systems actions so that explanation has access to all that has occurred and may summarize actions. Because all of the information is contained in the trace, explanation may occur long after the actions have occurred. Though the trace is generated as a side-effect and explanations of this trace are decoupled from the actual problem solving sequence, the traces themselves are impoverished. Execution traces only record what has occurred, not why. A common problem with direct trace-based attempts is that they lack a framework in which to interpret the contents of the trace prior to communication.

The explainable expert systems (EES) framework[12] bases explanation on knowledge structures that capture the rationale behind system actions. This is achieved by integrating explanation at the expert system specification and design phase, insuring that rationale knowledge is compiled into the expert system. In EES terms, at design time a *domain theory* that represented general problem solving strategies merges with a *domain-model*, modeling specific facts in the expert-system domain. The record of this is captured in a *design history*. This design history is useful in generating a more meaning rich execution trace.

To speak very broadly there are three phases to an explanation system's process: Being *Informed* about the subject of the explanation, *Interpretation* of such information, and finally *Generation* of communication that expresses the information to the user. Informing may consist of a sequence of assertions about a particular event simply building the representation that will be the subject of explanation. In the Interpretation phase, information is filled in to reflect a deeper understanding. Finally a generation component produces a sequence of communication actions that express relevant aspects of the interpreted information.

1.1.3 Natural language Generation

Natural language generation addresses the problem of producing natural language expressing pertinent facts from a given representation. Most natural language generation systems divide the problem into the three step process of determining the content of a generation, forming a plan to express this content, and realizing this plan as a sequence of words. The first two processes are termed the "strategic" side of NLG and the third is the "tactical."

The system TEXT[9] adopted a schema-based approach to strategic generation. Schema-based approaches use script-like structures that contain instructions on how to navigate a knowledge structure and produce a structured sequence. TEXT schemata integrated the navigation of the knowledgebase with the construction of the content plan. However schemata can be viewed as the result of a compilation process[11] where the rationale behind each instruction in a schema has been stripped away, limiting the degree to which explanations may be interacted with. In a planned approach[11] to strategic generation, a communication goal is mapped into an explanation plan through the application of plan operators. Subgoals within the text plan represent the rational behind the portion of the explanation underneath them. This enables the identification of the speaker's goal behind each portion of an explanation. As part of the EES project, Paris and Morre[11] designed a content planner that operationalized Rhetorical Structure Theory[8]. In this planner they recognized that navigation of the knowledgebase could be incorporated in the constraints of the plan operators and based explanations on a user model.

The tactical component of NLG concerns the realization of actual words to express content. Kay[7] proposed a declarative grammar formalism that allows for feature passing through unification in a phrase structure grammar. Most current realization systems use unification grammars as their underlying mechanism. A common use for such grammars is enforcing number, gender, and case agreement. There has been interest in integrating the strategic and tactical sides of natural language generation[6]. Many of the tasks that a content planner and a surface realizer do are analogous.

1.2 Comparisons To Previous Work

The approach here is similar to the Explainable Expert System (EES) Framework[12], however its model of query processing is built specifically to interpret CoBase operations. In addition we have developed a representation, classification and generation formalism that describes the complete explanation generation task. This formalism provides a structured approach to explanation, thus easing maintenance, integration, acquisition, and the interactive refinement and extension of machine-generated explanations.

1.3 CoBase Explanation Example

CoBase query relaxation will serve as the example here. Query relaxation is the process of weakening query constraints to increase the set of answers that satisfy a query. This is useful when a query does not return any answers, or the user is interested in exploring the semantic neighbors of the answers returned. Suppose the user poses a query seeking airports in Bizerte, Tunisia where runway lengths are greater than 5800'. It happens that there are no such airports, so CoBase relaxes the query from the location Bizerte to the region of Northeast Tunisia. After doing so CoBase finds a neighboring airport in the city Djedeida. Figure 1.1 appears.

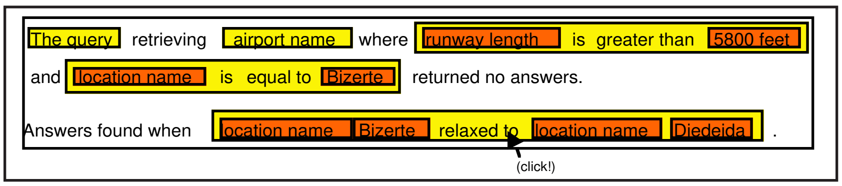


Figure 1.1: Message sent after user query was processed.

The explanation is brief. If the user requires more detail they interact with this initial explanation, obtaining further description. Sensitive text fragments are stacked hierarchically, letting the users interact with a single word, phrase, sentence, or paragraph. When the user "clicks" as above, figure 1.2 appears.

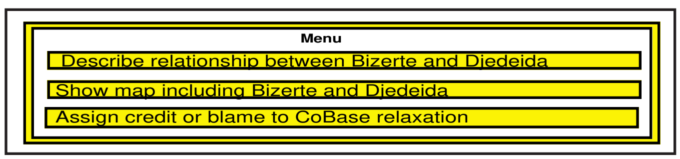
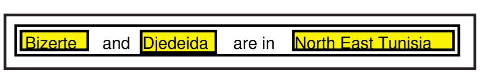


Figure 1.2: A menu presented to the user after the "click" of figure 1.1.

If the first option is clicked then figure 1.3 appears, and if the second option is clicked figure 1.4 appears. If the third option is clicked then the user is given the option of providing positive or negative feedback to refine CoBase's relaxation strategy. Such feedback is then absorbed into the CoBase relaxation control knowledge.



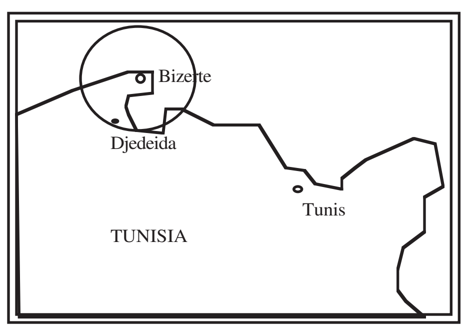


Figure 1.3

Figure 1.4

Although not illustrated in this example, explanation is user and context sensitive. Based on the user and context, explanations should occur at appropriate times during processing. One extreme is the system running automatically, only summarizing its work once it has completed. Another extreme is the system running in detail, explaining each action, giving the user the ability to monitor progress. Explanations should also be tailored to a user's understanding of the system.

Finally explanations must be extendible. CoBase is designed as a general system, capable of being applied to a wide range of domains. As such, an explanation system tied to CoBase must also be domain-independent. Though a domain-independent explanation may suffice, users prefer to have explanations as domain specific as possible. Hence it is important to have a facility that allows initially general CoBase explanations to be extended to more domain specific forms. For example the explanation in figure 1.1 could be better tailored to the transportation domain as in figure 1.5. Through a series of editing operations, the user transforms the initial, domain independent explanation into a domain specific form. In the future the explanation system will generate this explanation when working in the transportation domain.

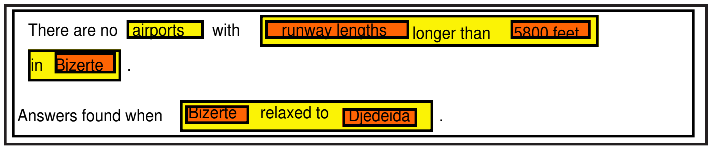


Figure 1.5: User improvement of explanation in figure 1.1.

1.4 Problem

CoBase must be able to provide descriptions of its queries and knowledge structures, real time descriptions of its actions, and finally descriptions of answers including explanations of how they were derived and their quality. These explanations should be interactive, user-sensitive, and context-sensitive. The explanation system must be a modular component, minimally impacting the design and implementation of other CoBase components. We have developed an explanation system which provides these capabilities for CoBase. The presentation of this explanation system will be the main focus of this report.

1.5 Organization of Report

Section 2 describes the system we have developed, giving an architecture and describing the input requirements and system process. Section 3 presents a formal model of this system including the representation of the subject of explanation and the explanation generation process. Section 4 describes our initial approaches to explanation refinement and extension and Section 5 gives conclusions.

2.0 System

2.1 Architecture

The explanation system is *capable* of providing multimedia, interactive, user-sensitive and context-dependent explanations of CoBase cooperative operations. The explanation system *requires* access to queries, TAHs[1], execution traces, and answers (Figure 2.1). A client sends the explanation system a simple *explanation request* on pieces of this information (e.g. describe query, explain answers) and receives back an *explanation reply* consisting of natural language and recommended visualizations that solve the request. The client presents the explanation on their GUI and a simple protocol enables the user to interact with initial explanations, causing the explanation system to generate follow up explanations. The explanation system maintains a model of query processing, consisting of concepts and classification rules,

[1]TAHs (Type Abstraction Hierarchies) are the principle knowledge structure in CoBase used to guide the query transformation process.

by which it interprets CoBase queries, TAHs, execution traces and answers. The explanation system applies generation rules to produce explanation replies from explanation requests.

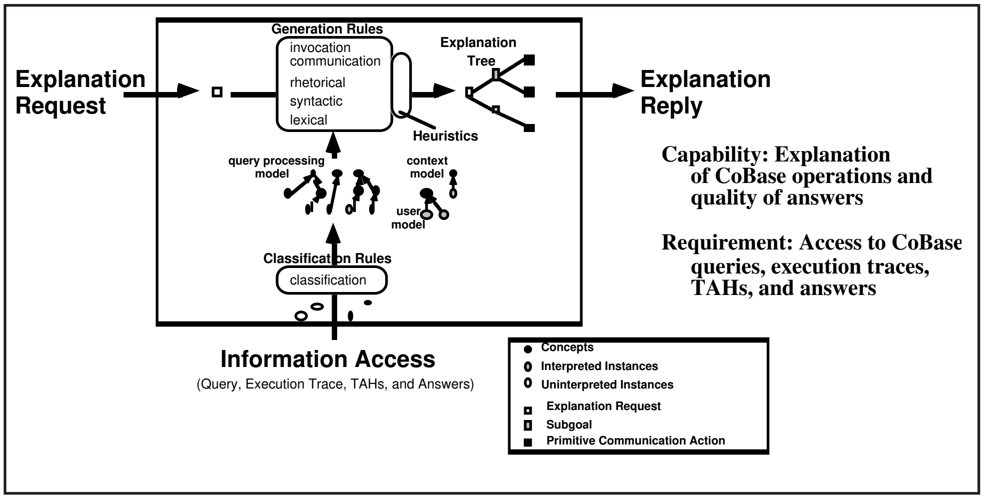


Figure 2.1: explanation system architecture.

2.2 Information Requirements

The explanation system represents information such as queries, execution traces, type abstraction hierarchies, and answers in a specific type of semantic network (Figure 2.2). The particular graph-based representation offers several advantages over standard semantic networks for explanation. Most importantly it allows for abstraction on the labels linking nodes (instances, values, or concepts), enabling specific to generic access of the graph[2]. The representation also enables the direct expression of one-to-many, many-to-one, and many-to-many relationships. Concepts in this graph (e.g. Query, Relax-Query-Action, etc.) include those of the query processing model . As the CoBase system executes it sends a stream of information that is instantiated in this graph. Classification rules in the query processing model interpret this information as it added to the graph. In addition the graph contains a user and context-model, enabling the generation of user-sensitive, context-dependent explanations.

[2]This property is not displayed on the graph in 2.2 for the sake of clarity. Section 3.1 illustrates this property for a portion of this graph.

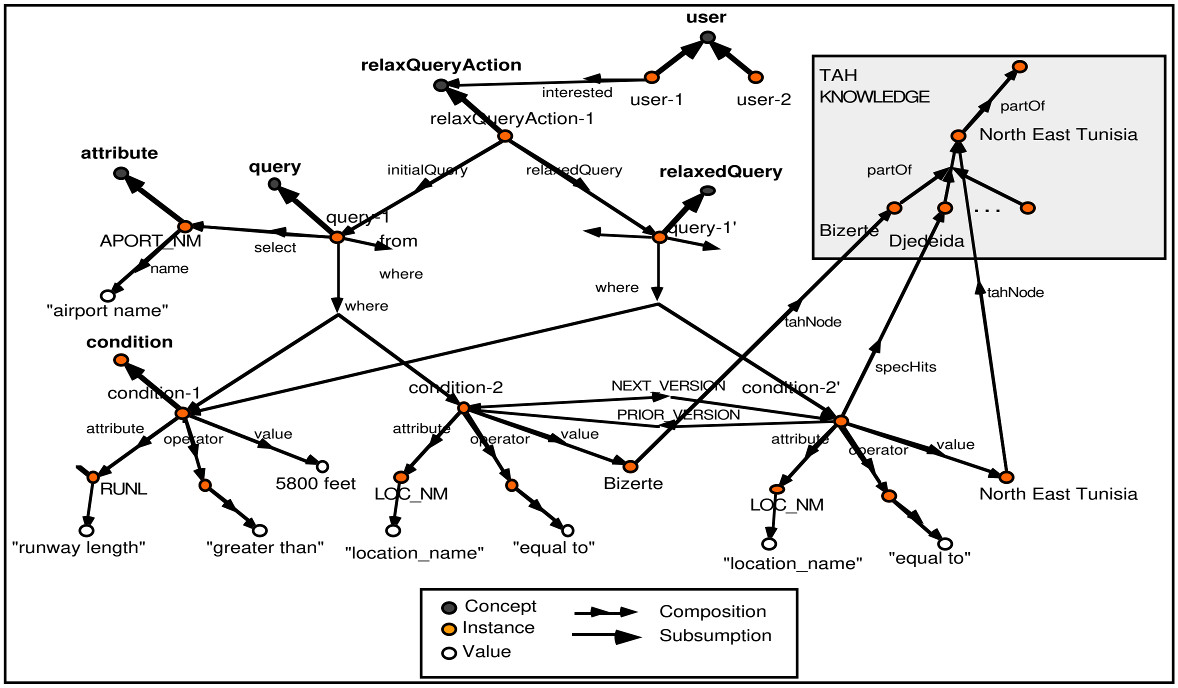


Figure 2.2: CoBase queries, execution trace, and TAH for the example of section 1.3.

Figure 2.2 shows an integrated view of the user query, query conditions and attributes, the relaxation process, the relaxed query, the user model, and the type abstraction hierarchy knowledge for the example of section 1.3. Specific discussion of the properties of our graph-based representation appear in section 3.1.

2.3 Explanation Generation

The explanation system accepts an explanation request on information in the graph and generates an *explanation tree* containing the explanation reply. This explanation tree is created via the depth-first application of generation rules toward solving the initial explanation request. The explanation request is at the root of the explanation tree, while the text and recommended visualizations are leaves in the tree. Intermediate nodes are the subgoals that are carried out to achieve the root explanation request. Generation rules consist of a goal, a set of constraints, and a set of actions. If the explanation request (or an intermediate subgoal) is matched by a generation rule goal, and if all of the rule's constraints are met, then the rule's actions are performed. These actions may either be subgoals that are in turn solved by application of generation rules or may be primitive communication actions (e.g. text or a visualization call). Rule constraints and actions access information in the graph as the explanation tree is being synthesized. A single inference mechanism applies classification and all levels of generation rules.

2.3.1 Classification Rules

Classification rules interpret CoBase information as instances of query processing concepts. For example assume a *relax-query* action is being traced and that at its conclusion if the resulting relaxed query still returns no answers then the relax-query action should be classified as a member of the concept *failed-relax-query* action. The paraphrase of such a classification rule would be: "If the relax-query action is complete and its resultant query has 0 answers then the relax-query action is a failed-relax-query action." All classification rules have this basic structure though the referent of a classification (e.g. the relax-query action) may be either an action or an object. Such interpretations of CoBase actions and objects are crucial to providing meaningful explanations. By performing such classifications inside the explanation system, the trace, query, and answer information being asserted by other CoBase components may be performed directly and efficiently.

2.3.2 Generation Rules

There are five types of generation rules: *invocation*, *communication*, *rhetorical*, *syntactic* and *lexical*.. Invocation rules determine what should be explained under various user models and contexts. For example in some cases users may wish to have their query described in English prior to its application. In other cases they may not. For example CoBase always issues the explanation request invoke after the query has been formed by the user. Which invocation rule is applied depends on the user and context. Say that the user is a novice using the system in a scrutinizing mode. A possible communication rule could be paraphrased as "If the form-query action is complete, the user is novice and the context is scrutinizing then describe the query produced by the form-query action." All invocation rules have this form and the referent of the invocation rules are exclusively CoBase actions.

Communication rules determine what information will be expressed about a topic in a description or explanation. Rhetorical rules express how that information will be presented. For example to paraphrasing the rule that describes an SQL query is "if a description of the query is required and if it is an sql -query then identify the query as an sql query, mention the attributes in the select clause of the query, and then describe all of the conditions of the query in order." The distinction between communication and rhetorical rules is not as clear cut as between other rules in this system. Syntactic rules constrain text to valid English. For example syntactic rules insure that subjects and verbs agree in number, or that items in a list are properly separated by ","s and "and"s. Finally lexical rules determine the words used to express CoBase actions and objects. For example such lexical rule might be to refer to the attribute "GLC_NM" as "location name."

2.3.3 Interaction

Users are able to interact to obtain deeper and more varied explanation beyond the initial explanation. These explanations may be further descriptions of graph information or they may be menus that users navigate to generate the explanation they require. These follow-up explanations are generated from reactive goals that are associated with nodes in explanation trees. When users interact with an explanation a reactive goal is issued to the explanation system. A follow-up explanation tree is generated from the reactive goal. In turn the follow-up explanation may be interacted with.

2.3.4 User and Context Dependence

Different users require different types of explanation under different contexts. This is accounted for by the fact that more than one generation rule may match to expand a node in the explanation tree. Alternate generation rules vary in which aspects of CoBase's process or results should be expressed and to what depth. Heuristic knowledge controls how an explanation is generated by controlling which rule among the matching rules will fire to expand a goal. Note that these heuristics apply only to generation rules which determine when and how to express information. The interpretation of the relaxation and association processes are non-heuristic. Through feedback and interaction users refine explanations by altering the heuristic knowledge that determines which rules are selected.

3.0 Formalism

The formal model of explanation generation consists of the representation that contains the subject of the explanation and a definition of the generation and classification mechanism. Upon this formal model of explanation generation an interaction model will be built that will enable user directed refinement and extension of generated explanations. Initial refinement and extension algorithms will be proposed.

3.1 Representation

The representation presented here captures the subject of explanation, the user model and the context model. This representation has several characteristics that differentiate it from standard semantic network representations. The most important is that there is a hierarchy among relationships that allow abstract access to a nodes neighbors. Such capability is crucial for later work on explanation acquisition and extension. When there are only very general rules available they must be able to traverse the graph through abstract labels. Another property in that this graph view may be treated as a hypergraph. This enables the direct representation of one-to-many, many-to-one, and many-to-many relationships.

3.1.1 Graph View

At its most basic level information is viewed as a simple directed graph G. Assume the following sets: V a set of vertices, $E \subseteq (V \times V)$ a set of directed edges, and Σ a set of values. The graph is $G = (V, E)$. A portion of the graph in Figure 2.2 is shown in figure 3.1.

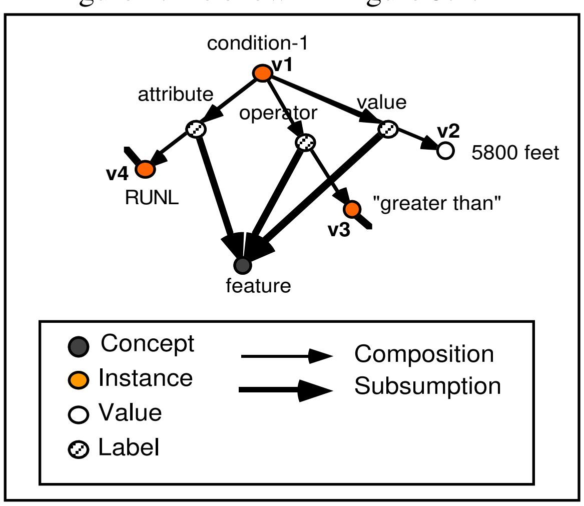


Figure 3.1: Portion of graph in Figure 2.2

There is a partial function Λ that partially maps the set of vertices to the set of values: $\Lambda(V) \rightarrow \Sigma$ (e.g. in Figure 3.1 $\Lambda(v1) =$ "condition-1". There are four types of vertices: concept, instance, value, and label. $V = V_c \cup V_i \cup V_v \cup V_l$. There are two classes of edges: subsumption and composition $E = E_S \cup E_C$. Subsumption edges represent *is-a* type relationships. Composition edges connect instance vertices to label vertices or connect label vertices to instance or value vertices. Such edges represent *property* or *part-of* type relationships. Finally there is a partial identification function I which maps identifier values to their corresponding concept or instance vertices. $I(\Sigma) \to V_c \cup V_i$.

For convenience a slightly more abstract view this graph is adopted. This graph may be viewed as a directed hypergraph. A hypergraph is simply a graph in which edges may be of arbitrary degree (normal graphs only have edges of degree 2). Figure 2.2 shows a hyper-graph. To help distinguish which view is being discussed, description of hyper graphs will be in terms of nodes and links. The label vertices along with their hierarchy have been excluded from this view, while concept, instance, and value nodes remain.

3.1.2 Graph Access

This section presents the mechanism to access information in the graph. The mechanism consists of two basic components: A single step neighbor access function S and a full graph access function T. Both of these mechanisms are discussed below.

Given a node in a graph, it is necessary to retrieve the set of neighboring nodes reachable by a given label value. The single step accessor function S takes a node and a label value as arguments and returns the set of neighboring nodes reachable by the given label: $S(V_c \cup V_i, \Sigma) \to 2^{V_c \cup V_i}$. On the graph in figure 3.1, S(v3,provider) = {v6}. S also may access neighboring nodes through abstract labels: S(v3,feature) = {v4,v5,v6}.

The full graph access function T uses the S access function as its primitive operator. T takes a node and a path expression and returns the set of reachable nodes via the provided path: $T(V,P) \to 2^V$. Path expressions are defined by the grammar: $P \to P \bullet P \| [or\ P] \| [P]^* | [P]^+ | [P]^\wedge | \sigma | e$. Informally • composes two path expressions, *or* bifurcates a path, * allows for any number of P, + allows for one or more P, ^ indicates the maximal number of P, $\sigma \in \Sigma$ is a single label and e is the empty expression. Ψ is a variant of T which nondeterministically returns a single reachable vertex: $\Psi(V,P) \to V$.

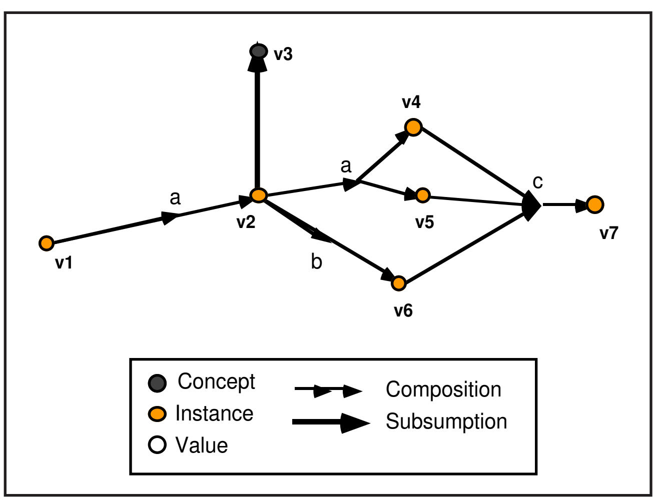


Figure 3.2: An example hypergraph

In the graph in figure 3.2: $T(v_1, \mathrm{a}) = \{v_2\}$, $T(v_1, [\mathrm{a} \bullet \mathrm{a}]) = \{v_4, v_5\}$, $T(v_1, [\mathrm{or}\ [\mathrm{a} \bullet \mathrm{a}]\ [\mathrm{a} \bullet \mathrm{b}]]) = \{v_4, v_5, v_6\}$, $T(v_1, [\mathrm{a}]^*) = \{v_1, v_2, v_4, v_5\}$, $T(v_1, [\mathrm{a}]^+) = \{v_2, v_4, v_5\}$, and $T(v_1, [\mathrm{a}]^\wedge) = \{v_4, v_5\}$.

3.1.3 Graph Synthesis

Either the subject of explanation must be asserted directly into the explanation system or the explanation system must have access to subject information. Typically small, unstructured information (e.g. Queries, execution traces, and answers, etc.) will be asserted directly while large structured pieces of information (e.g. Databases, Type Abstraction Hierarchies, etc.) will be accessed.

In the direct assertion option graphs are materialized through a sequence of basic operations which add vertices, add edges, and fill in the value function Λ and the Identification function I. Such basic operations are very low level so these operations are composed into higher level operations. The following simple assertion language presents four higher level operations.

Example Assertion Language:
define (concept [, concept-set])
instantiate (instance, concept-set)
fill-slot (node-set, node-set, value [,concept-set])
is-a (node,concept)
new (node-set)

Brackets **[]** indicate optional arguments. Sets are delimited by curly brackets **{}**. Singleton sets may be represented by their single element without brackets. This assertion language is complemented by a retraction language that offers reverses of these operations. Explanation must have access to the current as well as the prior versions of instances in the graph view. To achieve this the assertion language includes the operation *new*. The operation *new* is issued on a node just prior to recording changes to the node. New(x) causes: 1.) A new node *x'* to be generated. 2.) The identifier structure *I* is updated so that the identifier for *x* now points to *x'*. 3.) All of the links of the *x* are copied and added to *x'*. When these links are copied the nodes that they point to are made to be the newest versions. 4.) *x* is given a slot named "NEXT_VERSION" pointing *x'* and *x'* is given a slot named "PRIOR_VERSION" pointing to *x*. By using new a full history of instance and concept node versions is maintained in the graph.

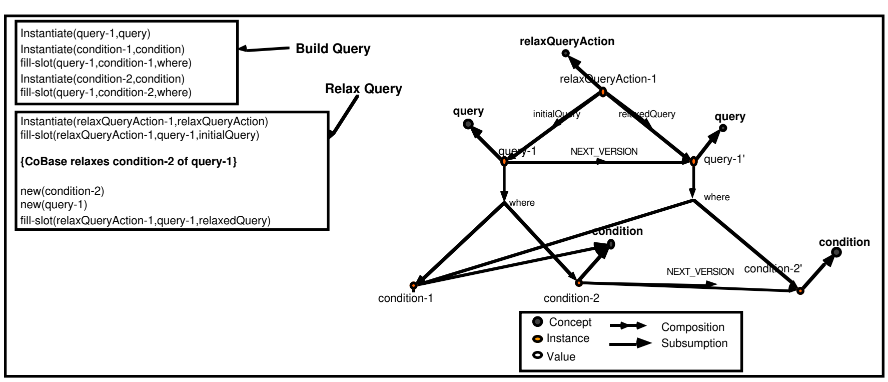


Figure 3.3: Assertions made by CoBase to generate execution trace.

Figure 3.3 shows some of the assertions made by CoBase to trace its execution in the example of section 1.3.

3.2 Explanation Generation

Explanations are generated by mapping explanation requests to explanation trees. Explanation requests are ground goal predicates (i.e. all arguments are bound) with arguments being keyword values or vertices (or sets of vertices) in the graph view. Explanation requests are mapped to explanation trees via the depth-first application of generation rules. This same mechanism is used to interpret information in the graph. The analog to an explanation request is a classification request which is issued on a vertex (or set of vertices) in the graph. A set of classification rules produce a tree similar to an explanation tree. However instead of having communication actions as the primitive actions, classification rules have assertions or retractions on the graph as their primitive actions. The explanation system does not return such trees to the client, but rather performs them itself, changing its graph to reflect the interpretation.

3.2.1 Generation and Classification Rules

Before we present the inference rules that define the generation and classification process, let us look at the structure of the rules. All generation and classification rules are of the form in figure 3.4

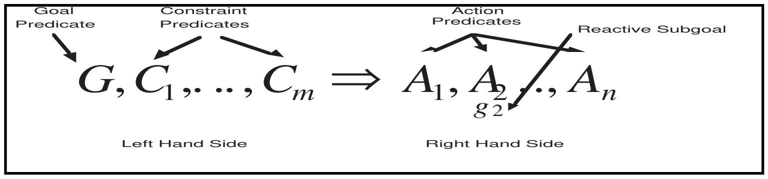


figure 3.4: classification and generation rule form

Goal predicates represent the purpose of the explanation rule. Constraint predicates access client information to verify the applicability of the rule. Action predicates perform primitive communication actions or pose subgoals. Reactive subgoals place interactive goals in the explanation tree. Such goals are solved if users interact with the corresponding portion of the explanation .

r1: classify(X),member(query,X), member(relaxedCondition,T(X,[where•[isa]*] =>
isa(X,relaxedQuery).

r2: invoke(X),member(Ψ(X,[isa]),T(USER,[isa]$^+$• interested)) =>
describe(X).

r3: describe(X), member(relaxQueryAction, T(X,[isa]$^+$)) =>
describe(Ψ(X,[initialQuery]), text("returned no answers"), text(" Answers found when"),
sequence(describe, relaxation-history,T(X,[relaxedQuery • where])).

r4: describe(X), member(query, T(X,[isa]$^+$)) =>
text("The query"), text("retrieving"), sequence(name,T(X,[select])), text("where"),
sequence(describe,T(X,[where])).

r5: describe(X),member(condition, T(X,[isa]$^+$)) =>
name(Ψ(X,[attribute])),text("is"),name(Ψ(X,[operator])), sequence(name,Ψ(X,[value]))).

r6: describe(relaxation-history, X), member(RelaxedCondition, T(X,[isa]$^+$)) =>
name(Ψ(X, attribute)), name(Ψ(X, [PRIOR_VERSION]^•value), text("relaxed to"),
sequence(name, T(X, [specHits])).

r7: describe(relaxation-history, X), member(Condition, T(X,[isa]$^+$)) =>
true().

r8: sequence(G,[X|S]), greater_than (Size(S),1) => G(X), text(","),sequence(G,S).

r9: sequence(G,[X|S]), equal(Size(S),1) => G(X), text("and"), sequence(G,S).

r10: sequence(G,[X|S]), equal(Size(S),0) => G(X).

r11: name(X) => text(X).

r12: name(X), exists(T(X,[name])) => text(Ψ(X,[name])).

Table 3.1: Example rules

3.2.2 Inference Rules

The process of producing an explanation tree from an explanation request will now be formalized. The definition is provided by a set of inference rules. In the following α and γ stand for sequences of predicates, the $*$ symbol divides the workspace stack from the output stack, Δ and ω are sequences of symbols on the output stack, $\mathrm{Eval(P)}$ evaluates the predicate P, θ is a most general unifier, σ is the rule selection predicate. *[r* and *]* indicate constituent boundaries on the output stack. Let explanation rules consist of a set of indexed predicates: $r_i{:}G_i, C_{i,1},\ldots,C_{i,n_i} \Rightarrow A_{i,1},\ldots,A_{i,m_i}$. A set of zero or more rejected rule identifiers is: $< r_{k_1},\ldots,r_{k_p} >$. These sequences record which rules have been tried and have failed on a particular goal.

[input]

$$G <> *$$

[match / select]

$$\frac{G < r_{k_1},\ldots,r_{k_p} > \alpha * \Delta \text{ where } \mathrm{G_i}\theta = G\theta \text{ and } \sigma(r_\mathrm{i},\{r_j | G_j\theta' = G\theta' \wedge \mathrm{Eval}(\mathrm{C}_{\mathrm{j,1..j,n_j}}\theta')\} - \{r_{k_1},\ldots,r_{k_p}\},\mathrm{u,c,\Delta,h})}{A_{i,1..i,m_i}\theta r_i G < r_{k_1},\ldots,r_{k_p} > \alpha * \Delta[r_i}$$

[fail / recover]

$$\frac{G < r_{k_1},\ldots,r_{k_p} > \gamma r G < r_{l_1},\ldots,r_{l_q} > \alpha * \Delta[r\omega \quad \text{where } \neg\exists \mathrm{G_i}\theta = G\theta \ \ \mathrm{i} \notin \{k_1,\ldots,k_p\} \wedge r \text{ not in } \omega}{G < r, r_{l_1},\ldots,r_{l_q} > \alpha * \Delta}$$

[fail / restart]

$$\frac{G < r_{k_1},\ldots,r_{k_p} > \gamma r G < r_{l_1},\ldots,r_{l_q} > \alpha * \Delta \quad \text{where } \neg\exists \mathrm{G_i}\theta = G\theta \ \ \mathrm{i} \notin \{k_1,\ldots,k_p\}}{G < r, r_{l_1},\ldots,r_{l_q} > \alpha * \Delta}$$

[evaluate primitive]

$$\frac{A\alpha * \Delta \quad \mathrm{Primitive(A)}}{\alpha * \Delta[\mathrm{Eval(A)}]}$$

[post goal]

$$\frac{A\alpha * \Delta \quad \neg\mathrm{Primitive(A)}}{\mathrm{Eval(A)} <> \alpha * \Delta}$$

[clear goal]

$$\frac{rG < r_{k_1},\ldots,r_{k_p} > \alpha * \Delta}{\alpha * \Delta]}$$

Table 3.2: Inference rules

The above inference rules perform a depth-first expansion of a top level goal presented by the **[input axiom]** inference rule. The end result of this process is an explanation tree on the output stack. The **[match/select]** inference rule selects a rule from all of the rules that match the goal and whose constraint predicates evaluate to true given the unifier on the goal. Rules that have previously been tried on the goal are filtered out before the selection occurs. It is here that the selection predicate is evaluated. In cases where all rules fail and backtracking is necessary, one of two inference rules are applied: The **[fail recover]** inference rule records rule failure with the parent goal and strips all of the associated action predicates of the failed rule from the output stack. The **[fail/restart]** ignores what was placed on the output stack and simply continues on. The **[post goal]** inference branches a subgoal for the current rule being evaluated. The **[evaluate primitive]** inference rule evaluates primitive actions on the right-hand sides of the current rule, leaving symbols on the output stack. **[Clear goal]** clears goals that have successfully been achieved.

3.2.3 Example

To illustrate the inference rules at work assume that the explanation request invoke(RelaxQueryAction-1) is issued with the graph from figure 3.2 with the rules from list 3.1 active. Through the application of inference formalism generation rules are applied to generate the explanation tree appearing on the output stack.

Inference	Input Workspace		Output Stack
[input]:	invoke(RelaxQueryAction-1)<>	*	
[match/select]:	describe(RelaxQueryAction-1)<> invoke...<>	*	[r2
[post goal]:	describe()<> invoke...<>	*	[r2
[match/select]:	describe(Ψ(RelaxQueryAction-1,[initialQuery])...<>	*	[r2 [r3
[post goal]:	describe(query-1)text(“ returned no answers”)...<>	*	[r2 [r3
[match/select]:	text(“The query”) ...<>	*	[r2 [r3 [r4
[evaluate prim]:	text(“retrieving”)...<>	*	[r2 [r3 [r4 “the query”
[evaluate prim]:	sequence(name,T(query-1,[select]))...<>	*	[r2 ... “the query” “retrieving”
[post goal]:	sequence(name,{APORT_NM})...<>	*	[r2 ... “the query” “retrieving”
[match/select]:	name(APORT_NM)...<>	*	[r2 [r3 [r4 ... [r10
[post goal]:	name(APORT_NM)...<>	*	[r2 [r3 [r4 ... [r10
[match/select]:	text(Ψ(APORT_NM,name))name(APROT_NM)<>...<>	*	[r2 [r3 [r4 ... [r10 [r12
[evaluate prim]:	name(APROT_NM)<>...<>	*	[r2 ... [r12 “airport name”
[clear goal]:	sequence(name,{APORT_NM})<>...<>	*	[r2 ... [r12 “airport name”]
• • •	• • •	*	• • •

Table 3.3 Example Inferences

3.2.4 Explanation Trees

The final bracketed material on the output stack represents the explanation tree. The first tree in Figure 3.4 shows the explanation tree corresponding to the output in the example above.

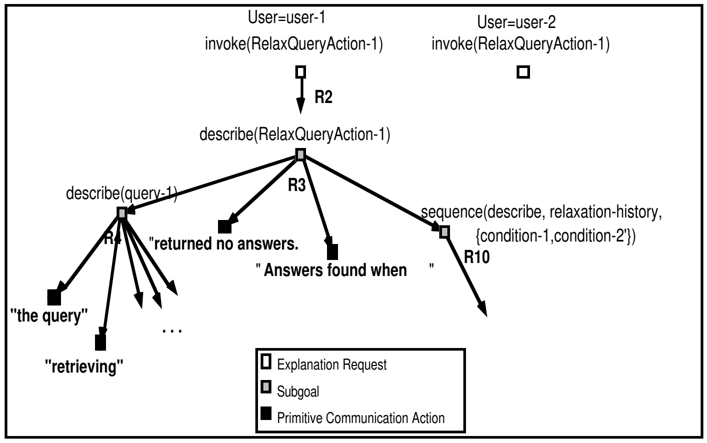


Figure 3.5: Example explanation tree

The other explanation tree consists of just a single node because the user was not interested in the topic of the top level invocation. An important relationship between the nodes in an explanation tree and the rules that generate it are that each leaf node is associated with one and only one rule predicate and each interior node is associated with rule that expanded it and the predicate of the rule that created it. The refinement and extension algorithms make use these relationships.

3.2.5 Selection

The predicate $\sigma(r, R, \mathrm{u}, \mathrm{c}, \Delta, \mathrm{h})$ in the [match select] inference rule controls which generation rule among the matching rules is used to expand a node in the explanation tree. *r* is the chosen rule from the match set *R*. *u* is the current user model and is bound to a single instance node in the user model graph. *c* is the current context and is likewise bound to a single instance node in the user graph. These user and context graphs are trees with the root representing all users or contexts and leaves representing specific users or contexts. Δ is the explanation tree context which captures the generation state prior to the selection of the rules. Finally *h* is the heuristic knowledge structure that is used to select the rule.

In the example in table 3.3 the twelfth inference selects rule r12 to expand the goal while both r12 and r11 match the goal. That is $\sigma(r_{12},\{r_{11},r_{12}\},u,c,[r_2\ r_3\ r_4\ r_{10}])$ is true while $\sigma(r_{11},\{r_{11},r_{12}\},u,c,[r_2\ r_3\ r_4\ r_{10}])$ is false . The explanation tree context can be summarized by listing the rules that have fired prior to the node that is being expanded (e.g. $[r_2\ r_3\ r_4\ r_{10}]$). The structure which controls which rules may be picked is the override graph h. The override graph for this example appears in figure 3.5.

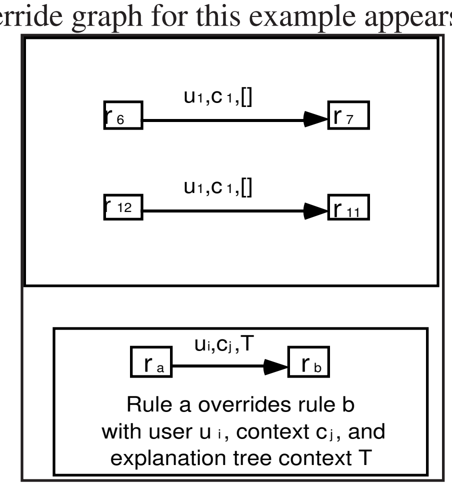


Figure 3.6: Example generation rule override graph.

The algorithm is to simply filter out overridden rules from the match set. The predicate $\sigma(r, R, \mathrm{u}, \mathrm{c}, \Delta, \mathrm{h})$ will evaluate true for the remaining rules and the [match/select] inference rule will non-deterministically select one of them. With u_1 representing all users and c_1 representing all contexts, in the current example it is clear that rule r_{11} will be filtered out, leaving only r_{12}.

3.3 Interaction

There are two modes in which a user may interact with an explanation: Investigative and Critical. In the investigative mode users are interested in obtaining additional explanation beyond the initial explanation. In the critical mode users provide feedback on the manner in which an explanation is expressed.

Investigative interaction is accomplished by clicking on a portion of an explanation. This results in a reactive explanation goal being solved and a follow-up explanation being presented to the user. Generation rules are responsible for placing reactive goals within the explanation tree through special action predicates. When users interact with an explanation, the minimal node within the explanation tree that covers the portion they have interacted with will be identified. The reactive goal associated with this node is then solved. If there is no associated reactive goal, then the nearest reactive goal on the path from this node to the root of the explanation tree is solved.

4.0 Further Research Issues

In addition to helping users obtain fuller explanation, interaction may be a means by which users critique explanations. Investigative interaction itself can be viewed as critique. If the user interacts with an initial explanation this can be view as a critique of the original explanation. The explanation was insufficient because the user required additional information. However, the focus here is on direct criticism of an explanation.

There are two types of criticism that can be leveled at an explanation: Inappropriate choice of expression and inadequate knowledge of how to explain. Section 4.1 treats the issue of refinement of how explanations are expressed and section 4.2 touches on the extension of explanations. Section 4.3 discusses the use of interactive explanation capabilities for CoBase knowledge acquisition.

4.1 Explanation Refinement

Inappropriate choice may be indicated by clicking on an improper portion of an explanation and cycling through all of the other possible choices. Once a better pattern is located the system can be informed of this. All of the patterns of expression that are passed over must then be overridden by the preferred pattern. This translates into introducing edges into the rule override graph. The user and context are always single nodes, determining the explanation tree context is the only complication. The explanation tree context is a sequence of the rules that fired prior to the improper portion of the explanation. Since only a portion of the prior context may be relevant, the interface tool will obtain from the user the scope of the refinement relative to the current explanation tree. With the rule override graph augmented, explanations will henceforth reflect the user's preference. The refinement algorithm is complete but requires exponential space and is sensitive to inconsistent interaction. We expect that the current efforts at evaluating this algorithm will determine a criteria to optimize the refinement work.

4.2 Explanation Extension

Inadequate knowledge of how to explain is a problem that needs further research. We do have an initial approach. After finding that a shortcoming in an explanation is not due to an inappropriate choice, a knowledgeable user must locate the closest match among the possibilities. After doing so the user performs a set of edit operations to revise the generated explanation. The set of operations are: change, delete, expand, or move. Users perform these operations until they transform the explanation into the form they wish. In addition to transforming the visible explanation, these operations transform and mark the explanation tree of the initial explanation. Once editing is complete, the transformed, marked explanation tree contains the new, domain specific explanation. This transformed explanation tree is traversed and when marked nodes are encountered new generation rules are induced. These new rules are induced from the original generation rules that expanded the transformed node in the initial explanation tree. These new generation rules are constrained to fire only when the current context and user model are active and the override graph is altered so that these new rules override the rules from which they were induced. By this process the explanation rulebase is populated with extensions to generation rules.

4.3 Explanation Assisted Knowledge Acquisition

Another direction to pursue is explanation-assisted knowledge acquisition. Though it acquires its major knowledge representations through discovery [2][10], CoBase needs to have a certain degree of manual knowledge acquisition. (e.g. naming TAH nodes). In addition if CoBase can explain its actions then the user has the capability to critique specific system decisions. CoBase could exploit this channel of communication to learn better control strategies. In other words the user perform credit assignment to CoBase strategies.

5.0 Conclusion

The explanation system has been implemented (in C++ and CLIPS) and integrates into the CoBase system. There are approximately 30 concepts and 20 classification rules in the query processing model. There are approximately 80 general generation rules and 40 domain specific rules for an electronic warfare domain and 30 rules for a transportation planning domain. The explanation system can usually produce explanations in under a second on a Sun Sparc 10 workstation. The quality of explanations (with respect to completeness, correctness, and precision) depends on access to CoBase queries, execution traces, TAHs, and answers, and also on the effort expended in populating and refining the explanation rulebase. Elapsed times depend on access cost to subject information and the computational complexity of synthesizing the explanation tree. The dominant cost is the synthesis of explanation trees, particularly the cost of matching explanation rules to expand nodes. A RETE[4] pattern-matcher performs these matches.

As CoBase extends it coverage of cooperative operations, the explanation system will be responsible for explaining more actions. The explanation refinement and extension approaches will be brought to bear on these tasks. Through this, we seek to further explore the synergies between explanation generation and cooperative information system technology.

References

[1] Chu, Wesley W. and Chen, Q. Neighborhood and Associative Query Answering. *Journal of Intelligent Information Systems*, 1(3/4), 1992.

[2] Chu, Wesley W. and Chiang, Kourong. Abstraction of High Level Concepts from Numerical Values in Databases. *Proceedings of the AAAI workshop on Knowledge Discovery in Databases,* July 1994.

[3] Chu, W., Yang, H., Chiang K., Minock, M., Chow, G. and Larson, C. CoBase: A Scalable and Extensible Cooperative Information System. Submitted to *Journal of Intelligent Information Systems*, September 1995.

[4] Forgy, C.L. RETE: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem*, Artificial Intelligence* 19, 1982, 17-38.

[5] Gaassterland, T., Godfrey, P. , and Minker, J. An Overview of Cooperative Answering. *Journal of Intelligent Information Systems*, 1(2):127-157, 1992. Invited paper.

[6] Kantrowitz, Mark and Bates, Joseph. 1992. *Integrated Natural Language Generation Systems* Aspects of Automated Natural Language Generation, Springer-Verlag.

[7] Kay, Martin. Functional Grammar. *Proceedings of the Fifth Annual Meeting of the Berkeley Linguistics Society,* pages 142-158, 1979.

[8] Mann, William and Thompson, Sandra. Rhetorical Structure Theory: a Theory of Text Organization. In The Structure of Discourse. Ablex Publishing Corporation, Norwood, New Jersey, 1987.

[9] McKeown, Kathleen R. Text Generation: Using Discourse Strategies and Focus Constraints to Generate Natural Language Text. *Cambridge University Press*, Cambridge, England, 1985.

[10] Merzbacher, Matthew and Chu, Wesley W. Pattern-based clustering for database attribute values*, Proceedings of the AAAI workshop on Knowledge Discovery in Databases*, 1993.

[11] Moore, Johanna D. and Paris, Cecile L. 1989. Planning Text for Advisory Dialogue. *In Proceeding of the Twenty-Seventh Annual Meeting of the Association for Computational Linguistics*, Vancouver, British Columbia.

[12] Swartout, William R., Paris, Cecile L., and Moore, Johanna D. Design For Explainable Expert Systems. *IEEE Expert*, Volume 6, Number 3, pages 58-64.