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

Abstract: Cooperative Information Systems provide approximate answers to a user's query when exact answers are unavailable and summary answers when answer sets are too large. Yet without an explanation of how and why such answers were derived, it is hard to estimate their usefulness. Further, the timing and level of detail of such explanations should be user and context dependent. In this paper, we present the architecture, representation, and process of an explanation system which provides such explanations of approximate answers. This explanation system has been implemented and integrated into the Cooperative Information System CoBase (UCLA) for a transportation planning and an electronic warfare domain. Our experience reveals that explanation generation is efficient and receives positive feedback from users.

Keywords: Intelligent Information Systems, Cooperative Information Systems, Explanation, Approximate Answers, Dialogue Management, Knowledge Integration, CoBase.

1.0 Introduction

Traditionally database systems require a user to understand the database schema and only return exact answers to a user's query. Cooperative Information Systems seek to enrich this conventional query-answer dialogue with the cooperative behaviors observed between humans[7]. Such systems allow a user to pose high-level or imprecise queries and to receive approximations when answer sets do not exist and summaries when answer sets are large. Yet such systems should also be able to explain how they interpreted a user's query and derived answers. In addition, a user should be able to interrogate the system to obtain a precise description of the actions it took in arriving at answers. Through this, a user may critique the techniques used by the system, so that the system is more cooperative in the future.

This paper addresses the use of explanation technology in the CoBase system. CoBase [1][3] is a cooperative information system, providing approximate and associated answers to a user's relational and object-oriented queries. This paper focuses on approximate query answering through query relaxation. Relaxation is 'cooperative' when a user is interested in approximate answers, which is often the case when exact answers are few or do not exist. CoBase uses the Type Abstraction Hierarchy (TAH) as its knowledge structure to guide query relaxation. High level nodes in the TAH represent general knowledge of a domain, while lower level nodes represent more specific knowledge. Terms and relations in the query are located in the TAH, are generalized, specialized, and then used to reformulate the query. TAHs can be generated automatically from the domain database through conceptual clustering algorithms[9][2]. CoBase emphasizes the scalability and maintainability of the TAH and the portability of the CoBase engine to different domains. CoBase's explanation system inherits these requirements.

The most common and rudimentary method to provide explanation is the placement of print commands within a source code or rulebase. Yet such approaches tie the generation of explanation too closely to the actual steps taken to solve the problem. As a result they explain things only as they happen, enabling only limited summarization of the systems actions. To uncouple system action and explanation, it is necessary to form a trace of the system's actions so that the explanation system has access to all that has occurred and may summarize actions and results. However in the absence of interpretation, these traces may not contain enough information to produce meaningful explanations. Execution traces only record what has occurred, not why. The Explainable Expert Systems (EES) framework[14] 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. Additional advances in explanation technology have resulted from improvements in how the actual explanation is generated. Initial template based approaches[13] produced one-shot explanations with which a user could not interact to obtain clarification or further description. Current methods tend to rely on a planned approach to generation [12], enabling interaction on portions of explanations.

Section 2 gives an example explanation for CoBase query relaxation. Section 3 presents an overall architecture of the explanation system. Section 4 focuses of how information to be explained is represented, section 5 discusses the explanation generation process, and section 6 compares our approach with previous work.

2.0 Example

Suppose the user poses the query:

Q: *Is there an airport in Gafsa, Tunisia with a runway length greater than 8500' ?'*

It happens that there is no such airport, so CoBase relaxes the location name attribute to find nearby airports with runway lengths greater than 8500'. The following explanation is presented.

There is no airport where
location is Gafsa and runway length is greater than 8500 feet.

There is no airport where
region is South West Tunisia and runway length is greater than 8500 feet.

Through the relaxation of Gafsa to South Tunisia an airport where
location name is gabes and runway length equal to 9126 feet has been found.

The whole region of Tunisia could still be searched for an airport where
runway length is greater than 8500 feet.

The user may select text fragments to obtain further explanation. Sensitive text fragments are stacked hierarchically, letting the user interact with a single word, phrase, sentence, or paragraph. For example, when the user "clicks" the text *'Through the relaxation of Gafsa to South Tunisia'* the following options are presented.

1.) Show map of South Tunisia including Gafsa and Gabes
2.) Give more detailed description of relaxation action
3.) Accept feedback to refine relaxation strategy

If the first option is clicked then a map appears, and if the second option is clicked then a more specific description of exactly how the query was relaxed is provided. If the third option is clicked then the user is given the option of providing positive or negative feedback on CoBase's strategy to relax location name over runway length for the current user and context. Such feedback is then absorbed into CoBase's relaxation control knowledge.

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. This is the case in the above example. Another extreme is the system explaining each of its actions as they occur, giving the user the ability to monitor progress. Explanations should also be tailored to a user's understanding of the system.

3.0 Architecture

CoBase's explanation system is *capable* of providing interactive, user-sensitive and context-dependent explanations and descriptions of CoBase operations. To provide such explanations, the system requires access to CoBase queries, execution traces, Type Abstraction Hierarchies (TAHs), and answers. The components of CoBase's explanation system (see figure 3.1) are 1.) a set of *generation rules* which produce explanations, 2.) a *query processing model*, by which CoBase queries, TAHs, execution traces and answers are interpreted (classified) , 3.) a *user model* and 4.) a *context model*. The user and context models affect which generation rules are used to produce an explanation. Calls to the explanation system either *provide information* or *produce explanations*.

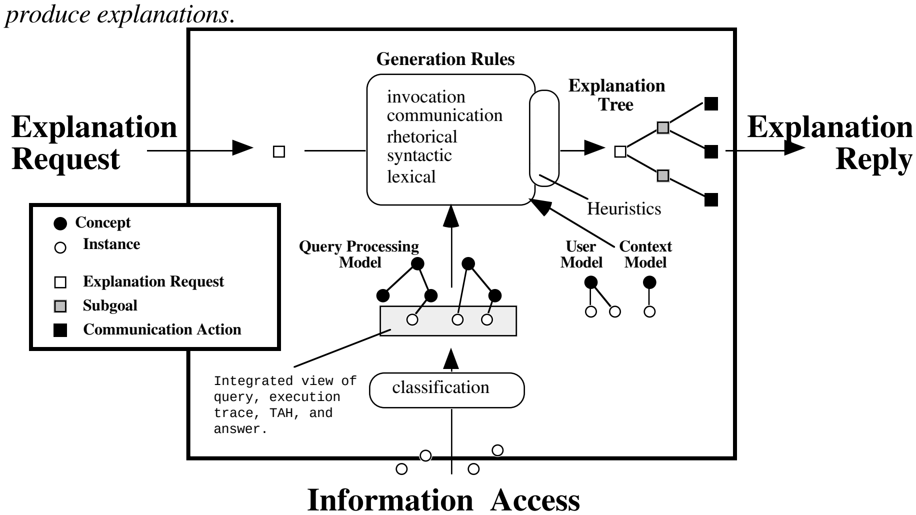

Figure 3.1 The architecture of CoBase's explanation system.

As CoBase executes, query, execution trace, and finally answer information is provided to the explanation system. Each piece of information is asserted as an instance of a concept in the query processing model. *Classification rules* may interpret instances as members of more specific concepts. The final result of these assertions and classifications is an integrated view of the query, execution trace, answer, and, by reference, TAH information.

To produce an explanation, the explanation system receives an *explanation request* on an instance in the integrated view (e.g. describe(query-01), explain(process-query-01, answers), etc.) and produces an *explanation reply* consisting of natural language and recommended visualizations. The explanation system solves explanation requests by applying generation rules to produce an *explanation tree*. Generation rules are applied top-down to the explanation request and resulting subgoals, producing the explanation tree where the explanation request is the root and primitive communication actions are the leaves. Generation rules access information in the integrated view and are often predicated upon the current user and context. Explanation trees are converted into a proper form (C++ objects, HTML, KQML, etc.) and are returned as explanation replies. These replies are presented on a GUI and a simple protocol enables interaction with the presented explanation, causing the explanation system to generate further explanation.

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 generation rules for a transportation planning domain. The explanation system can usually produce explanations in under a second on a Sun Sparc 10 workstation.

4.0 Representation of Information for Explanation

Figure 4.1 shows an integrated view of the query, execution trace, TAH, and answer for the example above.

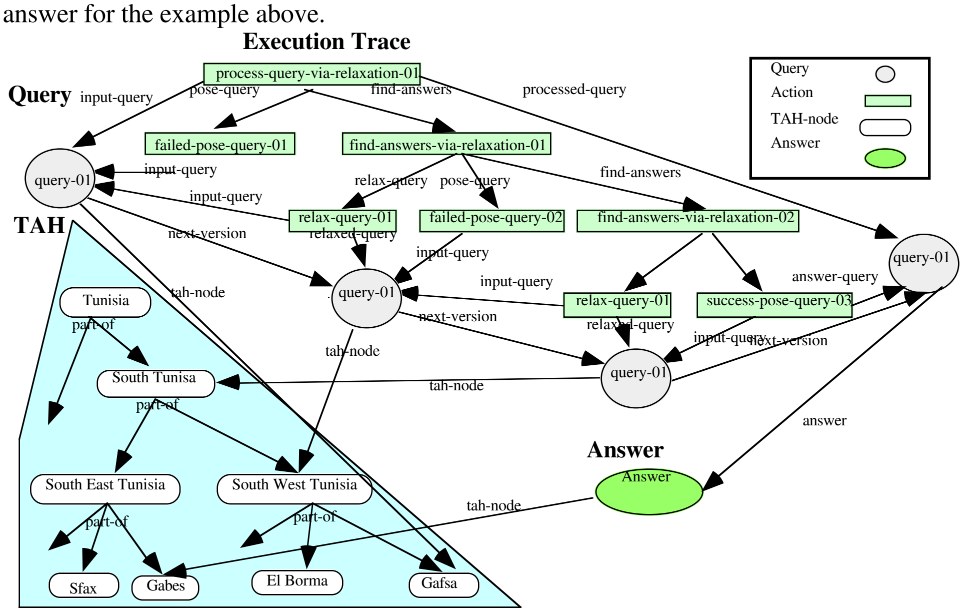

Figure 4.1 Integrated view of Query, Trace, TAH and answers for the example.

Classification rules in the query processing model interpret this information as it added. For example, the instance[1] "process-query-via-multiple-relaxations-01" in figure 4.1 was initially instantiated as a member of the concept "process-query-01". Later, after two relaxation actions, "process-query-01" was reclassified.

---

[1] Instance names reflect the most specific concept they are a member of. For example, "process-query-via-multiple-relaxations-01" is a member of concept, "process-query-via-multiple-relaxations".

4.1 Basic Graph View

The information in figure 4.1 is presented at a conceptual level. Figure 4.2 shows how the explanation system actually represents this information. For reasons of space, figure 4.2 is limited to only a portion of the query and execution trace in figure 4.1.

At its most basic level, information is viewed as a simple directed graph G. Assume the following sets: V a set of vertices, E a set of directed edges, and Σ a set of values. The graph is G=(V,E). There is a partial function Λ that maps the set of vertices to the set of values: $\Lambda(V)\rightarrow\Sigma$ (e.g. in Figure 4.2 $\Lambda(v_1)$ = "condition-1"). This has the effect of naming each vertex. There are four types of vertices: concept V_c, instance V_i, value V_v, and link V_l. Each Concept and Instance vertex has a unique identifier: i.e. for $\Lambda(V_c \cup V_i)\rightarrow\Sigma$, Λ is complete and one to one. For convenience we define the partial identification function I which maps identifier values to their corresponding concept or instance vertices: $I(\Sigma)\rightarrow V_c \cup V_i$ where $I(\sigma) = \Lambda^{-1}(\sigma)$ for $\sigma\in\Sigma$. There are two classes of edges: subsumption E_s and composition 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.

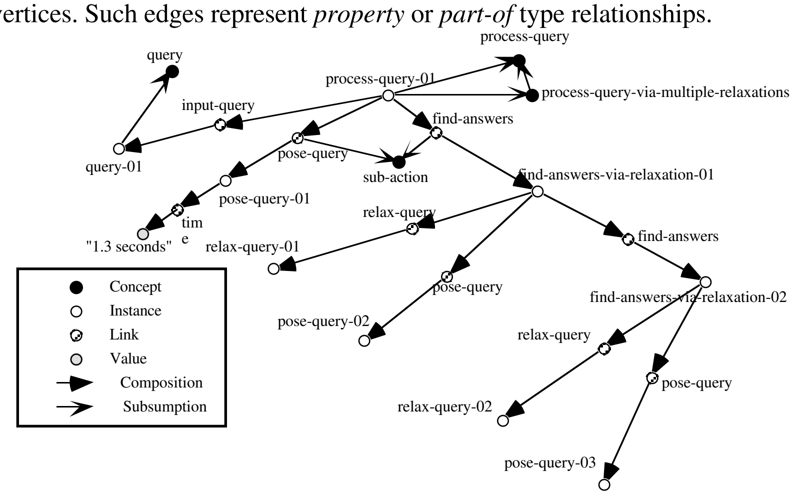

Figure 4.2 Graph portion of execution trace and query in figure 4.1.

4.2 Neighbor Access

The mechanism to access information in the graph consists of two functions: A single step neighbor access function S and a full graph access function T (discussed in section 4.4). For S, given a node in a graph, it is necessary to retrieve the set of neighboring nodes reachable by a given link value. The single step accessor function S takes a concept or instance vertex and a link value as arguments and returns the set of neighboring nodes reachable by the given link: $S(V_c \cup V_i,\Sigma)\rightarrow 2^V$. On the graph in figure 4.2, S(I(process-query-01),input-query) = {I(query-01)}. S also may access neighboring nodes through abstract labels: S(I(process-query-01),sub-action) = {I(pose-query-01),I(find-answers-via-relaxation-01)}.

4.3 Hyper-Graph View

For convenience, the basic graph may be viewed as a directed hyper-graph. In such a view edges may be of arbitrary degree (normal graphs only have edges of degree 2). The hyper-graph is constructed from the basic graph by retaining all concept, instance, and value vertices and introducing hyper-edges with label σ from v to V' where S(v,σ) = V'. To help distinguish between the basic or hyper-graph view, description of hyper graphs will be in terms of nodes and relationships rather than vertices and edges. Figure 4.3 shows the hyper-graph view of the basic graph in figure 4.2.

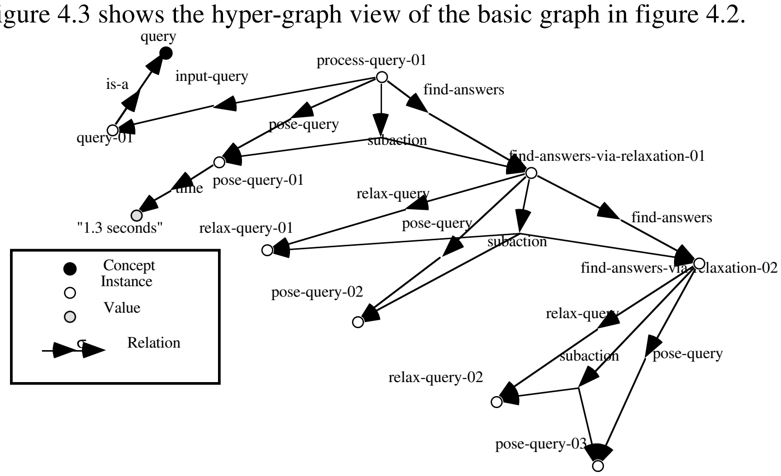

Figure 4.3 Hyper-graph view of basic graph in figure 4.2.

4.4 Full Graph Access

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: $T(V,P)\rightarrow 2^V$. Path expressions are defined inductively by the grammar

$$P \rightarrow e \mid \sigma \mid P \bullet P \mid P^* \mid P^+ \mid P^\wedge \mid P^n \mid or[P\ P] \mid and[P\ P]$$

where $\sigma \in \Sigma$ is a single label, *n* is an integer and *e* is the empty expression. Intuitively • composes two path expressions, * allows for any number of P, + allows for one or more P, ^ indicates the maximal number of P, P^n allows for n or fewer P's, *or* follows both paths, and *and* finds those node only reachable by both sub-paths. Ψ is a variant of T which nondeterministically returns a single reachable node: $\Psi(V,P)\rightarrow V$. We have developed a formal definition of T and Ψ. For the graph in figure 4.3, T(I(process-query-01),input-query)={I(query-01)}, T(I(process-query-01),find-answer*•pose-query)= {I(pose-query-02),I(pose-query-03)}.

4.5 Graph Synthesis

Either information must be asserted directly into the explanation system or the explanation system must have access to information. Typically smaller, unstructured pieces of 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.

Graphs are materialized through a sequence of basic operations which create or destroy vertices and edges in the simple graph view, effecting the value(Λ) and identification(I) functions accordingly. Such operations have been grouped to form a higher level assertion language consisting of five basic operations defined on the hyper-graph: define(concept), is-a (node, concept), instantiate(instance, concept), fill-slot(node, node, slot) and add-value(node, value, slot), This assertion language is complemented by a retraction language that offers reverses of these operations.

To maintain a trace, access to current as well as prior versions of nodes must be maintained. To achieve this the operations new-shallow (node), and new-deep (node) are added to the assertion language. These operation are issued on nodes just prior to recording changes to the node. The operation new-shallow creates a new copy of the node and attaches the new and old copy with the slots "next-version" and "prior-version". The operation new-deep performs the same operations as new-shallow, but in addition, the slots from the older node are copied to the new node and updated. By using these operations, a full history of instance and concept nodes is maintained. We have developed a formal definition of these operations.

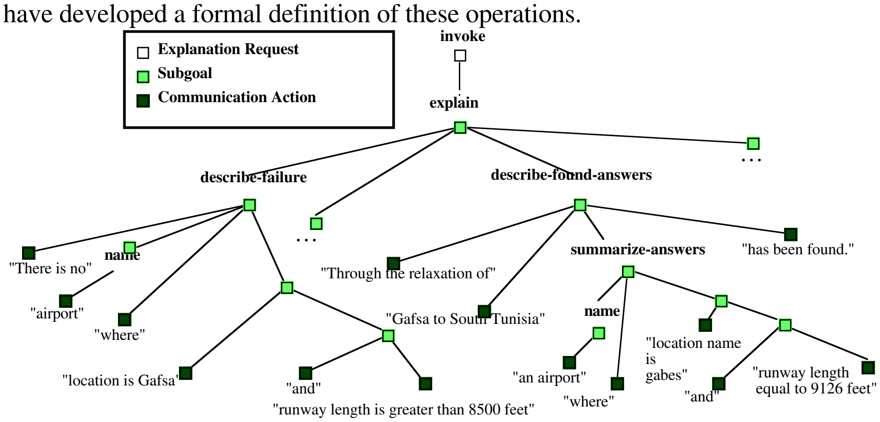

Figure 5.1 Explanation tree for the example

5.0 Explanation Generation Process

With the method for representing, accessing, and asserting information established, we now turn to the process of generating actual explanations. The explanation system accepts an explanation request on information in the graph and generates an explanation tree containing the explanation reply. Explanation requests are ground goal predicates (i.e. all arguments are bound) with arguments being keyword values or nodes (or sets of nodes) in the graph. An explanation tree is created via the depth-first application of generation rules toward solving an 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. We have defined this process formally[10]. Figure 5.1 shows the explanation tree generated for the example.

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 node (or set of nodes) in the graph. A set of classification rules produce a tree similar to an explanation tree. However classification rules perform assertions or retractions on the graph as their primitive actions. The explanation system does not return these trees as replies, but rather performs them itself, reflectiong an interpretation in the graph. Such interpretation is important to providing meaningful, specific explanations. By performing such classifications inside the explanation system, the query, execution trace, and answer information may be asserted directly and efficiently.

All generation and classification rules are of the form as shown in figure 5.2. Goal predicates represent the purpose of the rule. Constraint predicates access information in the graph. If all the constraints are satisfied then the rule's actions may, based on heuristics, be performed. These actions may either be subgoals or primitive communication actions (e.g. text or a visualization call), or, within classification rules, assertions or retractions on the graph.

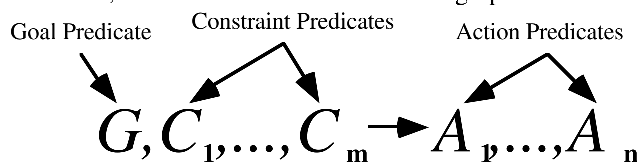

Figure 5.2 The form of classification and generation rules

5.1 Classification Rules

Classification rules interpret instances as members of more specific concepts. For example, assume a *process-query* action is being traced and that due to the failure of the initial query, approximate answers are sought through relaxation of the query. The process-query action should then be classified under the more specific action *process-query-via-relaxation*. The following classification rule performs this interpretation:

(r12 classify(X), member(process-query,T(X,[isa]*)),
exists(T(X, find-answers•relax-query)) → isa(X, process-query-via-relaxation))

5.2 Generation Rules

There are five types of generation rules: *invocation*, *communication*, *rhetorical*, *syntactic* and *lexical*. *Invocation rules* determine which actions should be explained under various user models and contexts. For example a user may wish to have their query described in English prior to CoBase's attempt to solve the query. In other cases a user may not wish for such a description. CoBase always issues the explanation request *invoke* on the *form-query* action after the query has been formed by the user. The following is an example of a general invocation rule that states that "If an action is of interest to the current user, then describe the action."

(r34 invoke(X),member(Ψ(X,[isa]),T(USER,[isa]$^+$•interested)) → describe(X))

Communication rules determine what information will be expressed about a topic in a description or explanation. *Rhetorical rules* capture how that information will be presented. For example, if an explanation of the entire *process-query* action is required, and if the process-query action required relaxations, then describe the failure of the original query and then describe found answers

(r23 describe(X), member(process-query-via-relaxation, T(X,[isa]+))→
describe(Ψ(X,pose-query),failure),describe(Ψ(X,find-answers^)),found-answers))

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 name CoBase actions and objects. For example the following are general lexical rules.

(r19 name(X) → text(Λ(X)))

(r20 name(X), exists(T(X,[name])) → text(Λ(Ψ(X,[name]))))

5.3 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 are expressed and to what depth. Heuristic knowledge controls which generation rule among the matching rules will fire to expand a goal. Through feedback and interaction users refine explanations by altering the heuristic knowledge that determines which rules are selected.

Because user and context models will not be completely accurate, users may interact with an initial explanation to obtain further explanation. These further explanations are generated by issuing explanation requests on nodes in the explanation trees. The solution of explanation requests on explanation tree nodes results in a new explanation tree containing the further explanation. In turn this explanation may be interacted with.

6.0 Comparison to Previous Work

Few Cooperative Information Systems[5] have included an explanation facility. McCoy [8] generated explanations to clear up a user's misconceptions of a database schema. The CARMIN [6] system includes an integrated explanation and answer presentation system. In this system explanations are based on the proof path used by a PROLOG meta-interpreter. Aspects of what to include and how to coordinate these explanations were also investigated. In contrast to these approaches, the work here is focused on explanation of approximate answers. In addition our explanation system is interactive, giving the users the capability to fully inspect and critique the query relaxation process.

Similar to EES, information is interpreted prior to generation of explanation. However a model of query processing is built specifically to interpret CoBase operations and trace-generating operations are added to existing CoBase routines. Though this implies that the CoBase engine and the query processing model are maintained separately, this modularizes the explanation system and reduces its impact on the design and implementation of other CoBase components.

We have developed a representation, classification and generation formalism that describes the complete explanation generation task. The particular representation enables the direct expression of one-to-many, many-to-one, and many-to-many relationships and allows for abstraction on the relations between nodes (instances, values, or concepts), enabling specific to generic access of the graph. Abstraction enables very general rules to access the graph thought abstracts labels. The generation formalism provides a structured approach to explanation, thus easing maintenance, integration, and in the future the acquisition, interactive refinement and extension of explanations.

7.0 Conclusion

The completeness, correctness, and precision of explanations depend on access to queries, traces, knowledge, and answers, and dende on the effort expended in populating and refining the explanation rulebase. Elapsed times depend primarily on the cost of synthesizing the explanation tree, 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. This will include explanation of associated answers to a user's query and explanation of summary (intensional) answers. Efforts will also focus on methods to enable users to interactively refine the explanations they receive [11]. In addition, since explanations are based on the domain-independent CoBase engine, we seek a method that enables system administrators to interactively extend explanations into domain specific forms[11]. Through these experiments, we seek to further explore the benefits of explanation generation for 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. To appear in *Journal of Intelligent Information Systems*, 1996.

[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.

[6] Godfrey, P., Minker, J., Novik, L. An Architecture for a Cooperative Database Systems. *International Conference on Applications of Databases,* June 1994.

[7] Grice, H. Logic and Conversation. In P. Cole and J. Morgan, editors, *Syntax and Semantics*. Academic Press, 1975.

[8] McCoy. *Generating Natural Language Text in Response to Questions About Database Queries*. PhD thesis, University of Pennsylvania, 1982.

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

[10] Minock, Michael J. and Chu, Wesley W. Interactive Explanation for Cooperative Information Systems. *UCLA Computer Science Department Technical Report*, SD950036, 1995.

[11] Minock, Michael J. and Chu, Wesley W. Generation, Refinement, and Extension of Explanations for Cooperative Information Systems. In *Proceedings of CIKM Workshop on New Paradigms in Information Visualization and Manipulation,* December 1995.

[12] 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.

[13] Shortliffe, Edward H. *Computer Based Medical Consultations: MYCIN*. Elsevier North Holland Inc., 1976.

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