1

MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS
STANDARD REFERENCE MATERIAL 1010a
(ANSI and ISO TEST CHART No. 2)

**National Library
of Canada**

**Bibliothèque nationale
du Canada**

Canadian Theses Service

Service des thèses canadiennes

**Ottawa, Canada
K1A 0N4**

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

Using Case-Based Reasoning for Knowledge-Based Assistance

by

Una-May O'Reilly

**A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of
Master of Computer Science**

**School of Computer Science
Carleton University, Ottawa, Ontario**

August 28, 1990

© copyright 1990, Una-May O'Reilly

Government of Canada
of Canada

Gouvernement du Canada
du Canada

Canadian Theses Service Service des thèses canadiennes

Ottawa, Canada
K1A 0N4

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

ISBN 0-315-60481-6

**The undersigned recommend to the Faculty of Graduate Studies
and Research acceptance of the thesis**

Using Case-Based Reasoning for Knowledge-Based Assistance

**submitted by Una-May O'Reilly, BSc
in partial fulfillment of the requirements for
the degree of Master Of Computer Science**

Thesis Supervisor

A handwritten signature in black ink, appearing to read "Frey Oppacher". A handwritten signature in black ink, appearing to read "F. Fiala".

Chairman, School of Computer Science

Carleton University

August 1, 1990

ABSTRACT

Case-Based Reasoning (CBR) is a paradigm for memory-based mental processing. A case-based system uses a library of episodes as its knowledge base and formulates a solution to a new problem by retrieving the most similar situation from its case library and adapting its solution to use it for the new problem. Case-based systems are better than rule-based systems because their knowledge is easier to acquire, can be extended easily, and is more like the natural processes experts use.

This thesis makes the following contributions to Case-Based Reasoning:

- Graphs are used to represent cases. They are a flexible representation which supports the formulation of different interpretations according to situational context.
- CBR is used in a novel manner as a paradigm to provide knowledge-based assistance (KBA). Traditional CBR systems perform classification or planning.
- KBA in design domains has been accomplished using CBR. Traditional rule-based methods are inadequate to capture the unstructured, unconstrained, creative nature of these domains.
- Domain experts are used as 'oracles' which places only minimal reliance on them to be articulate and to initially supply the system with a large body of correct and precisely formulated knowledge.

A novel approach to automated knowledge acquisition and machine learning facilitates a CBR system which can:

- handle incomplete cases as input.
- learn and refine heuristics using an initially small amount of approximate knowledge.
- influence the match quality of retrieved cases by reasoning about context and purpose.
- adjust its performance at the usually fixed stages of CBR, (analysis , matching and using a case) by employing feedback.
- extract a 'weak' theory of how to create good designs.

The extensions are illustrated with examples from a prototype system, CMA, which provides KBA for an Entity-Relationship Diagram design tool.

**This thesis is dedicated to
my Grandfather,
George Eric Marthins
(1905 - 1989)**

ACKNOWLEDGEMENTS

I extend my thanks to Franz Oppacher who has guided this research with loose reins in terms of schedule but a firm grip in terms of influencing its direction. There were often many choices and Franz encouraged me to examine their implications and select the ones which would be the most stimulating and challenging. He helped greatly by showing me how to formulate my fuzzy ideas more clearly and by often realizing that I had more than I thought I did. His friendship has been exciting and I am looking forward to pursuing my Doctorate under his supervision.

I owe three fellow graduate students, Dwight Deugo, Alan Knight, and Andrew Rau-Chaplin, a great deal of thanks. They provided supportive, argumentative, challenging, and never boring friendship. Thanks also to Paul White, Suzy Skublics, Rob Black, and the School of Computer Science support staff.

The support of Bell Northern Research through their Post Graduate Award Program is gratefully acknowledged. In particular, I'd like to thank Dr. Jack Dymment and members of the University Interaction Group for their time and assistance.

This work got part of its impetus from the AMOS project. I would like to thank COGNOS Inc, for their contribution. The people to be thanked are Stan Matwin, Jean Genest, Zbig Koperzak, and Bertrand Pelletier.

My family is last so their acknowledgement may be remembered as the most important. My parents, Glenda and Bob, have loved me without question through 28 years and I would never have believed in myself without my husband's never failing confidence in me. Thank-you Stephen Yee. Pet Barney and Pawla for me.

TABLE OF CONTENTS

1 Introduction	1
1.1 Organization.....	13
2. An Introduction to Case-Based Reasoning	15
2.1 The Motivation for CBR.....	15
2.2 Memory and Reminding	19
2.3 CBR versus Rule-Based Reasoning	24
2.4 An Overview of the CBR Process.....	25
2.5 Case Representation.....	27
2.6 Steps in the CBR Process	30
2.7 Summary	40
3. Case Representation	44
3.1 Issues in the Representation of Cases.....	44
3.2 MODELLER and CMA.....	50
3.3 Using Graphs To Represent Cases.....	52
3.3.1 The Merits of Graphs in CMA	57
3.3.2 Using Features Associated with Graphs	63
3.4 Summary	69
4. CBR and Knowledge-Based Assistance	71
4.1 CMA: An Assistant for MODELLER.....	72
4.2 General Knowledge-Based Assistance Issues	76
4.3 CBR and KBA.....	78
5. An Extended Model Of CBR	82
5.1 The Extended Model of CMA.....	83
5.2 Generating Suggestions	83
5.3 Describing and Using Heuristics	92
5.4 Training the System.....	94
5.5 Cases and Input.....	100
5.6 Knowledge Acquisition.....	101
5.7 Discussion of the Processing Extensions in CMA	105
5.8 Concept Description.....	109
5.9 What the System Does	111
6. Future Research and Summary.....	112
6.1 Future Research	112
6.2 Summary	113
A.1 A Description of CMA's Operation	116
References	125

LIST OF TABLES

TABLE	DESCRIPTION	
Table 2.1	Comparison of CBR Systems	41

LIST OF FIGURES

FIGURE	DESCRIPTION	
Figure 1.1	The Case-Based Reasoning Process.....	3
Figure 1.2	Theory Extraction	7
Figure 2.1	RPD MODEL.....	18
Figure 2.2	The CBR Model.....	27
Figure 3.1	E-R Diagram Example.....	50
Figure 3.3	An E-R diagram as a Graph.....	55
Figure 3.4	CBR Shell.....	56
Figure 3.6	Graph Data.....	59
Figure 3.7	MODELLER Example in Graph Representation (partial)	60
Figure 3.8	Subgraphs in a Graph	61
Figure 3.9	Connected Components	64
Figure 3.10	Cut Points.....	64
Figure 5.1	Generating a Suggestion	64
Figure 5.2	Using the Expert as an Oracle	64
Figure 5.3	The Feedback Loop in CMA.....	64

LIST OF APPENDICES

Appendix A.1 A Description of CMA's Operation	116
--	------------

1 INTRODUCTION

The motivation of this thesis has been to design an extended model of case-based reasoning (CBR) which can provide knowledge-based assistance. We focus our attention on applications where the cognitive task is a design process and an expert requires help while using a tool to construct a good design. A primary goal of such a system is to track the type of help a user wants and provide assistance in that specific aspect.

The system's knowledge is employed to provide its user with plausible suggestions for completing the current design. Expert designers rely heavily on what they learned through creating other designs. Specific examples from the past greatly influence the new designs they generate. Recognizing this, our design is from the case-based reasoning (CBR) paradigm. Our knowledge base is a (case) library of finalized designs. We obtain a system that not only puts memory to profitable use but which is relieved of requiring a detailed causal model of the application and not restrictively modelled upon any design methodology. The system inherits advantages of the CBR paradigm: faster reasoning with experiential justification and simple knowledge acquisition.

To achieve successful use of CBR for KBA, innovations and extensions to CBR are needed. Graphs are successfully used in a novel way to represent cases. They are advantageous because they facilitate different informational views of data without requiring representational changes. CBR systems are more powerful when they make decisions on their own that influence their performance rather than relying solely on expert supplied decisions. For instance, a system which can decide on its own what makes an episode similar to another is more robust than one that is given its similarity assessment criteria a priori. By aiming for a system which is responsive to demands of a particular user we concentrate on improving this aspect of CBR. We propose a training mechanism which involves the user giving simple feedback to the system so it can tune its own performance.

Typically in knowledge-based systems, generality is achieved through an overall architecture which separates domain specific knowledge from the general purpose reasoning components. The supplemental information structures (heuristics) and modifications to the CBR algorithm we present influence only the general purpose reasoning components and thus maintain generality.

The domain we present for illustration is Entity-Relationship modelling for database design. The issues of this application have given rise to the previously described overall goals of the design and are factors in more subtle decisions we describe later. An implementation of an assistant for a tool called MODELLER, named CMA, serves to demonstrate the feasibility of our design.

The expert systems of the 1970's and 1980's modelled sophisticated behaviour and decision making by using the mechanisms of deductive and inductive inference over rules combined with knowledge bases of facts. Such systems are effective because this is a cognitively plausible model of some types of human problem solving. One particular facet which such a model neglects, however, is the notion of memory and learning from experience. The so-called knowledge acquisition bottleneck and knowledge-base "brittleness" arise from the inadequacy of the model to represent information in episodic form so that it can be re-used without having to be derived all over again, and, from the inability of the systems to dynamically re-organize the knowledge they contain to incorporate new knowledge as each new experience is used to ascertain more about the domain. In addition, rules do not capture the depth of knowledge in the domain which expresses the reasons for decisions and the intrinsic meaning of the actions.

A case-based reasoning system is one which is modelled around the notion of memory and reminding for the purposes of understanding, planning and learning. So much of the mental processing people do depends upon memory. Memory and the process of being reminded are central factors in peoples' ability to cope with new encounters and continually learn from them. The cognitive advantage people have over less intellectually advanced beings is this ability to learn about their environment and to develop the skills to adapt to its constant changing. Memory is an essential ingredient to their success; once work has been done - cognitive work involving assessment, planning, failure correction and experimentation, it is rarely all forgotten. Some patterns of knowledge and the cues to instantly retrieve and put them to use again are retained. People are relieved of always having to figure something out again, they simply remember the solution they already derived. We will later present psychological evidence which substantiates this claim.

A case-based model is shown in Figure 1.1. CBR is a relatively new approach to learning and understanding. Pivotal early work on the memory and case-based paradigm is attributable to Roger Schank and Patrick Winston in the 1980's [Schank82, Winston80]. The first extensive results were published in 1986 [Hammond86a,

Hammond86b, Kolodner84, Kolodner86, Kolodner87]. Case-based reasoning contributes solutions to the problems of a knowledge acquisition bottleneck and the knowledge base brittleness in assistant systems. This is because cases are problem solving episodes which are easily captured in the course of experience and a case library eliminates the need for a deeply structured explicit theory of the domain. In areas of expertise where a causal theory is absent or experts are unable to articulate their knowledge in terms of rules or consistent problem solving methodology a case is the best source of information which can be drawn upon.

The thesis includes a description of the components of a case-based reasoning system and explains how they function together to form a general mechanism of the memory, reminding, and learning paradigm. An overview of the significant contributions which guided the evolution of CBR to its current capabilities and open problems [Barletta88, Deugo89, Hammond88, Hammond89a, Rissland89, Sycara88] is presented as a context to the primary motivation of this thesis: the extension of the general CBR mechanism to use it for intelligent assistance and to facilitate the system making flexible decisions which improve its own performance.

Figure 1.1 The Case-Based Reasoning Process

Central to a case-based system is a library of cases. The case library represents memory. These are episodes which describe the factual descriptions and conditions under which actions were taken. An episode captures the context of an entire situation. It may have characterizations in terms of plans or goals which are relevant or be a set of functionally driven specifications. The library itself can be a flat assembly of cases or support organization which expresses the relationships among cases in a hierarchy or discrimination network. The most important thing a CBR system must do is to retrieve the case which is the best one to use as a basis of solving the new problem. Without a good quality exemplar, all of the further reasoning which is based upon examination of this example becomes inconsequential.

The CBR system initially processes its input problem by recognizing salient features with which it can retrieve partially matching episodes from its library. With knowledge of the significance and priority of alternative matches, the "best" match is chosen. Domain independent knowledge or domain theory is then employed to adapt the actions prescribed by the best match to suit the input conditions. The system is able to anticipate conflicts which could disrupt the success of the solution because episodes have their unusual circumstances marked so they can be avoided later. When the system's actions are used the results are interpreted (which may result in an update of the library information) and incorporated into the system. Often failure is the outcome, but this is can be handled in a positive manner when the reasons are understood and incorporated back into the case library. Thus if there is something new to be learned the system acquires it from its experience.

This thesis contains a discussion of memory and schemes for its representation in case-based systems. Representations of the memory need to be suitable for a design process and suitable for case-based reasoning. The merits and disadvantages of a rigidly structured (frame-based) representation for a case are covered. The knowledge representation of CMA's cases is, instead, graph based. We propose that case representation by graphs is a better way to represent cases because it can adequately express the objects and relationships of a wide variety of domains and because it facilitates flexible interpretations of data as different types of information.

CBR is an obvious paradigm for providing assistance in completing design tasks. As will be shown later, experts rely as much on their experience creating previous designs as on any formal theory. Completed designs can constitute the case library. However, a completed design or case does not express design theory but is instead an intermediate form from which it is derivable. In many domains - E-R design being one; it is

impossible to formalize a design theory by any means (not just examining cases but even by scrutinizing an expert's activities) without it being subjectively concocted. We need mechanisms which, objectively, by evidence which is experiential in nature, extract the less formal (weak) theory which exists inside a case so it can be used within the case-based paradigm.

Instead of presupposing an explicit description of design theory for its domain, we propose priming the system with a wide range of initial interpretations (heuristics) of assessment, matching and helping. Using the subjective, qualitative and inarticulate judgement of an expert the system, during a training phase, discriminates amongst the initial interpretations and adjusts their definitions to approximate an experientially justifiable formulation of the design theory for its domain.

The heuristics start as preliminary hypothetical definitions of general modelling theory applied to this specific domain. Then they become formulations of a weak application-specific theory. For example, the derivation of the concept 'complete' in this domain may mean a high density relationship between entities, relationships and specifications of cardinality. While a retro-analysis via generalization from a case back to a structured general modelling theory is possible, pragmatically (and in the purest notion of CBR) we circumvent or ignore this. After all, ultimately the user wants help expressed in terms of his application. The system reasons, not by relying on theory, but on experience to get quick, inferentially short-circuited approximate information.

The design theory is expressed by 3 general notions: 1) knowing what a design needs in the next step of construction (assessment of incomplete designs/models) 2) knowing what makes one completed design like another in terms of the design theory that might be extracted from it and knowing why one match of design to incomplete design is better than another for the same purpose (similarity factors and metrics) 3) knowing what specific suggestion to offer (assistance generation). This is expressed by Figure 1.2.

Our enhancement to the general data structures of a CBR system is to define two classes of heuristics. An importance-based heuristic is a predicate which if verified, results in a belief by the system, defined by its meaning, with a given significance. A parameter-based heuristic is one which defines its predicate in terms of a test over a range of acceptable values. The parameter specification facilitates fine grained expression of a belief held by the system.

We extend the processing algorithm of a CBR system, in a general manner, not only to use the beliefs determined by instances of these two classes of heuristics, but to actually encompass a training phase during which the system itself acquires the strength

values of its importance-based heuristics and discovers the parameter bounds of its parameter-based heuristics. The processing algorithm is modified to support a feedback loop which permits the expert acting as an 'oracle'¹ to influence the system's self adaptive use of its belief structures so that those which are important to guiding the system to provide its best responses become higher priorities. The oracle simply indicates which results are satisfactory, inconsequential or poor.

There are arguments in the CBR community about what level is most appropriate upon which to base similarity. The merit of shallow features which simply exhibit syntactic knowledge is that they are automatically at hand and thus offer quick results. Abstract (deep) features need to be derived from the surface characteristics of a case and thus require more effort and background knowledge to obtain. They do, however, express deeper information with regard to causality or explanation. The domain is often the crucial factor in this choice because ultimately the best way to characterize knowledge is dictated by what its use is. What is often ignored however, is how the identification of the features is decided, not, whether features will be deep or shallow. It is more important that a system can acquire definitions for similarity criteria by itself rather than rely on the a priori specification of them and that it not require hand coded methods of identifying features. Our mechanism and heuristics are a facility for automatic feature acquisition so they circumvent the issue of deciding a priori what is best for a given domain and they eliminate the need to explicitly define similarity metrics for a domain.

We believe that a case-based reasoning model is an excellent basis for providing knowledge-based design assistance. Knowledge-based assistant systems can be thought of as computer coaches which act as advisors to persons tackling a set of tasks with a computer or as intelligent tutorial assistants which assist pupils in learning. In Chapter 4 we state the reasons why CBR is suitable and briefly emphasize the CBR issues which arise from the using CBR to provide KBA. These reasons and issues, which are now summarized, have motivated our CBR extensions.

When it comes to sharing the knowledge, that is, trying to help another person with the task being pursued, memory is a primary resource. The system or person lending assistance will need to draw upon memory in order to give the best guidance possible. A good helper can empathize with the person who needs help because he knows

¹Oracle (def, Oxford Dict): The answer of a god or the inspired priest to an inquiry made respecting some affair. The seekers of such divine communication or message were left to interpret the underlying meaning as it pertained to their appeal. The oracle provided no reason or explanation.

Figure 1.2 Theory Extraction

that this "sticky" situation is similar to ones other people get into and he knows of a previous solution which may work now because it has similar features and the same types

that this "sticky" situation is similar to ones other people get into and he knows of a previous solution which may work now because it has similar features and the same types of complicating interactions. The ability to anticipate how an old episode needs to be changed to work this time is also stored within memory as part of the episode or captured in some set of generalizations which can be accessed and used. In a sort of Samaritan attitude, one can save someone else the agony and exertion of repeating the steps that led to failure and which required backtracking or reversal. Advice saves work for the advised and when good advice originates from a memory, mental work has been saved by both the helper and the receiver of the help.

Teachers provide better assistance to their pupils as they learn more about their task of teaching and about the subject area they are teaching. Because a case-based system learns from its experience, its ability to assist improves as it evolves. The concepts of memory updating and re-organization in a CBR system enable learning to take place so that the capabilities of a systems are continually strengthened.

Sensitive helpers recognizes the capability of their pupil and try to provide help which is at an appropriate level to the pupil's skills. Thus, a system analyst may assume the functions of the accounting department and the purpose of a general ledger before recommending a specific relationship which might exist from the former to the latter. A teacher making the assumption that the pupil knows fundamentals is complemented by a case-based system which short cuts deduction and inference and uses memory as a quick substitute. The fundamentals do not have to be re-iterated. A case-based assistant system does not have to use the basic, detailed sequence of steps to derive a suggestion. This also obviates the need to laboriously process a detailed solution to tailor it for presentation in a form which is appropriate to the user's skill level.

Sensitivity to the user's needs, in terms of detail and level of assistance, in an assistant system is achieved by maintaining a current model of the user. Much work in user modelling in assistant or dialog systems focuses on the systems tracking the user's beliefs and intentions [Chin89, Kobsa89, Quilici89]. Tracking goals presupposes an explicit theory of the application because plan recognition and generation along with goal recognition rely upon knowledge which defines the choices, expresses the reasons for the choices and specifies what potential interactions exist. Since cases can remove much of the burden of a system requiring an explicit detailed causal representation (i.e. 'deep') representation of the domain, it is also plausible that information in cases can be used by a user modelling component. Cases capture goal definition and plan generation as well as

being templates for plan recognition. A case which is judged to match the user's current scenario may be beneficial for user modelling.

A KBA system is supposed to be unobtrusive and diligently all-knowing. It should not interrupt its user to determine why something is being done or what particular step in a general solution a particular sequence of actions represents. Furthermore, it acts like a coach, proposing suggestions as its user continues to work. While an overall goal may be known to the system, it may not know which plan has been selected to meet the goal and it must assume that the user may be building his plan incrementally, or "on the fly" as he receives feedback from some subjective personal criteria.

For example, in the process of entity-relationship modelling for database schema design an analyst is involved in a verbal exchange with the client and, after a briefing session, starts to create a model from the imperfect model in his mind. As the model is created by the analyst in front of the client, the client provides more detailed information and the analyst refines both the mental model and the model in front of both of them.

The issue for a case-based assistant system in this context is how it should appraise both the information currently available and the general top-level goal to use the features of this incomplete task description to find a completed task which is similar. An general extension to the CBR model which tackles this issue, input appraisal is presented in Chapter 5.

Applying CBR to KBA also yields another issue. The problem is that the goals and features of the input are incomplete when suggestions are required while the case library contains cases which are complete solutions. This gives rise to a requirement that the knowledge representation of these cases be in a decomposable form so that smaller sub-cases can be recognized and used as similarity criteria when case retrieval and matching is performed. The representation must also communicate the temporal ordering of tasks and multi-factor constraints which influenced the sequence. In Chapter 3 graphs are shown to be a representation of cases which meets these demands.

Because cases are 'captured' complete episodes, they not only satisfy the demands of the domain, but they additionally contain personalizations, that is, stamps of subjective idiosyncrasies. These idiosyncrasies are misleading, in one sense, because they are not influenced by rules or behavioral necessities. Sometimes several options may have been available and one was chosen because of habit, personal vagary or syntactic convenience for one person.

For example, when analysts build entity-relationship models they can start by first creating many entities then, next, proceed to define the relationships between the entities.

selects the first option is that he finds it easier to complete all mouse chores before all keyboard commands instead of interleaving the two activities then this choice is not cognitively valuable and in some fashion the freedom to differ in this respect must be represented.

Unfortunately, (or to complicate matters), if cases are provided by an expert they may contain some idiosyncratic choices which are excellent examples of high quality decisions and actions for which the expert has become admired and renowned as an expert. They may capture lateral thinking or be products of split second personal mental cues thus drawing upon the wealth of experience every expert accumulates.

It is possible for the knowledge engineer and the expert who provides the cases to examine cases and attempt to identify personal factors within them. These factors can then be distinguished from the absolutely necessary actions which satisfy the constraints of the application and then sub-divided as due to expertise or idiosyncrasy. Such an exercise is neither simple nor short. It can turn into a soul-searching, brain-picking, second-guessing routine in which too much is asked of the expert. Not only may his recollection fail him, but he may not be able to articulate why he made certain choices or to decide whether an unconstrained choice was simply habit or a significant semantic or syntactic decision.

When more than one system analyst examines an E-R diagram case these experts will even disagree about whether the reason a decision was taken is "important" to helping someone or simply a subjective choice. Timing the knowledge engineering and acquisition tasks to take place prior to system development so that such information discrimination can be embedded into the knowledge base results in inflexibility in the system and limits the generality of its case-based model processing modules. The system is inflexible because should the environment under which cases are derived change (as most real ones do) or experience noisy stimulus then the discrimination performed on the knowledge in the cases may not be adequate for new ones. Nor does the system have the capability to adapt itself to identify any new discrimination factors and to put this discernment to use in its processing algorithm.

Another aspect of this flexibility requirement relates to the precept that an assistant system is considered effective by its user if it gives him the help he wants (not necessarily the best help according to the system!). For such a requirement a KBA system using cases needs to find in its cases the information a particular user wants. It is not sufficient that even the team composed of knowledge engineer, expert and a user panel define, in general, what a user wants. There should be a certain amount of system personalization for one user within the general framework. Since this system capability relies on

examining the knowledge for its pertinent information, if it were to be done for each user prior to system development then the development work would be expensive, limited for re-use and too specific. This thesis presents a learning mechanism which fits into the general processing model of a CBR system so that the system itself learns what aspects of the domain expertise are important to the user. The learning mechanism permits a system to adapt itself:

1. to use an increasingly appropriate appraisal criteria for assessing its input when a user asks for a suggestion so that it can focus what is needed most.
2. to recognize the important features in matching situations and the parameters which define these features for matching according to the current state of the model and results of appraisal.
3. to determine what in the best matched case is important information to use as the basis for a suggestion.

Since the user is the most important judge of the quality of an assistant system, this thesis proposes that a training phase involving the user is an advantageous extension to the evolution and definition of a case-based assistant system. Involving the user permits him to actually influence the direction in which the system focuses so that he will be satisfied. The user participates with the system, which is initially run in a development mode, using his judgement to provide the learning mechanism with feedback which the system uses to tune its self-adaptation.

In CHEF [Hammond89b], a CBR system which generates new recipes, repairs are made after the user submits a failure report. A planner by Deugo and Oppacher [Deugo89, Oppacher88] handles failure and replanning "on the fly". We suggest that a training phase can use both failure directed and success directed learning to prepare a system. A training phase suits the notion of a learning curve where initially a great deal of learning is done and later it tapers off. People learn best when they are told what they have done well in addition to what they have done wrong. A training phase with positive and negative feedback choices can exploit this idea.

This thesis will relate the results of a prototype implementation, Conceptual MODELLER Assistant (CMA), which is a knowledge-based assistant that employs case-based reasoning as the "engine" and model of its knowledge-based processing.

Conceptually CMA is the encapsulated assistant component adjoining another knowledge-based tool called MODELLER.² The context for MODELLER is as follows.

MODELLER is a conceptual blackboard that system analysts and their clients use to build an Entity-Relationship (E-R) data model of the clients' management information systems. It is mainly used to document the logical properties of a database (DB) during the DB design process. The model represents the entire enterprise's view of data and is an abstraction independent of storage and efficiency considerations. It is mapped to an appropriate DB schema which is actually realized by some data base management system (DBMS). The intention is that an enterprise schema is stable under different user views and even with changes to the underlying DBMS. The basic structures are entities - the objects, and relationships - the time specific behavioral interactions in a system. An Entity-Relationship Diagram, (E-R diagram), is a diagrammatic depiction which illustrates entities as rectangles and relationships as diamonds between directed arcs. An E-R data model is a compromise of orthogonal requirements - generality for semantic richness and practicality to permit schema implementation in one of the commercial DBMS.

MODELLER is itself an expert system which relieves the expert analyst of performing certain consistency checks and common transformations which are needed before an E-R design can be converted to an actual physical schema. MODELLER executes logical to physical model conversion automatically and provides a data dictionary which can be directly used by a specific DBMS.

The important aspect of MODELLER relating to assistance is that it is very much a tool for experienced and knowledgeable system analysts. While it is extremely powerful and relieves them of certain conceptual and working tasks and provides a smoother interface for the analysis sessions the systems analyst still has a sophisticated set of mental tasks to perform because the E-R data modelling process itself is very sophisticated. CMA is intended to assist the user of MODELLER with the cognitive demands of the modelling process.

CMA is implemented in Smalltalk: V-286. Its software is sizable. The classes and methods are divided into the case-based processing shell, the domain specific component, the graph application, and interface utilities. The graph application contains the smallest amount of code, 100K bytes, and comprises 15 top-level classes with 5 subclasses. It handles the representational definition and manipulation of cases. The domain specific component interfaces the specific aspects of MODELLER with CMA. It has 100K bytes

² MODELLER is the proprietary property of COGNOS inc.

of code and comprises 7 top-level classes with 20 subclasses. The case-based shell implements the processing driver and related sub-components. While it is the largest module, its classes and methods encapsulate the general aspects of the case-based reasoner. Since this is the most portable part of a CBR application comprising the general purpose capabilities the 300K bytes of code and 20 top-level classes with 15 subclasses are merited.

CMA is a vehicle through which we, as artificial intelligence researchers investigating learning, can focus on the issues of using case-based reasoning for knowledge-based assistance. It has been designed without the stringent demands of producing robust production level software quality nor trying to satisfy the wide spectrum of customer delivery requirements that are elemental to commercial expectations. In designing CMA - its extended case-based engine which self-adapts by training, and its application specific modules, then by implementing key components, both the complementary facets of the assistant and CBR combination and the gaps between the goals of assistance and the known facilities of CBR became apparent. These will be discussed in Chapter 5. We will also use CMA to illustrate issues described in the literature of CBR. It is a concrete example which highlights some of the subtle and sophisticated fine-points of CBR.

We will present examples of the importance-based and parameter-determination heuristics devised for general CBR system training which we used to reveal information about the E-R diagram design process and which could not be readily obtained by asking MODELLER's most expert users. These heuristics can extend the system's effectiveness in a way which is not always anticipatable and provide a way of circumventing the very real problem of dealing with an expert who can only imperfectly articulate what he knows about his domain. The system, though only a prototype, furnishes evidence that CBR is not only an adequate processing paradigm for assistant systems but with regard to the aspects of knowledge acquisition, flexibility, generality and satisfaction of quick response demands, a CBR paradigm furnishes excellent solutions.

1.1 ORGANIZATION

This thesis proceeds as follows: Chapter 2 presents psychological evidence that CBR is a process used by experts and reviews the processes of reminding. It then presents a detailed overview of the CBR processing model which is supplemented with examples from CBR systems in the literature. It concludes by stating the weaknesses with

current systems and the opportunities in CBR that should be exploited and extended in future research. Chapter 3 discusses case representation. It starts by stating the requirements of an adequate case representation for design domains. We then explain why graphs are a good way to represent cases, using examples from CMA to illustrate the advantages. In Chapter 4, knowledge-based assistance (KBA) and CBR are considered together. We discuss why CBR is a suitable paradigm on which to base KBA and what requirements arise in designing such a system. Our extended model of CBR is motivated by its use for KBA and as a result we can also investigate some of the as yet unresolved issues in CBR which were described in Chapter 2. Chapter 5 is a description of the extended model of CBR which we propose. It includes examples and discusses how the performance of the model compares with established CBR techniques. Chapter 6 presents future work and a summary. Appendix A.1 is an annotated example of CMA.

2. AN INTRODUCTION TO CASE-BASED REASONING

In this chapter we start by presenting psychological evidence supporting case-based reasoning as a plausible cognitive model. We discuss what knowledge structures are appropriate to function as memory and analyse reminding for what it reveals about how memory must be structured in order to support retrieval of the most closely related case and its own re-structuring to incorporate newly learned knowledge. Comparisons with rule-based systems highlight the advantages inherent in CBR.

We then completely describe CBR systems in terms of the general algorithm and knowledge structures that may be employed to implement memory-based processing. This description draws upon documentation of existing CBR systems. We also use this opportunity to expose the unexplored and unresolved issues of CBR. The chapter ends with attention drawn to those aspects which were considered in this thesis.

2.1 THE MOTIVATION FOR CBR

The purpose of reasoning is to understand the events which take place in our lives and to provide support for the decisions we take. How do we reason? And, how do we learn? Without argument, there are many ways. Both rigorous science and common sense tells us that our memory is crucial for both learning about and understanding the world.

In order to embrace case-based reasoning one must be convinced that a memory based processing system is a plausible model of human reasoning. Without doing this, it is possible to admire the techniques and results of current CBR systems but one would fail to recognize the enormous potential which exists for CBR systems of the future. Current systems flaunt with the primitives of learning and understanding from memory. They achieve adequate results in restricted applications. When these principles are better understood, the range of use for a CBR system will increase and more opportunities to exploit the power of its learning model will be at hand.

Experiments have been conducted to examine if there exists psychological evidence of people using analogues and to determine the purpose of analogues [Klein88a, Klein88b]. The specific goal of this work was to determine "what functions are served by analogical reasoning during planning and decision making?" [Klein88a, p209]. While

the use of analogues for creative purposes in special cases had been confirmed by anecdotal reports of scientists and mathematicians, evidence of analogy use in a non-extraordinary context was being sought. The researchers reviewed a data base of protocols describing over 400 decisions made by experienced decision makers performing a variety of tasks. Four domains were considered; urban firefighting, wildland firefighting, tank platoons, and system design engineering. In each domain it was obvious that the decision maker had occasion to retrieve analogues and use them as a basis for decision making.

For example, a design engineer had to determine if a 60 degree field-of-view display would allow effective training. His users wanted a more expensive display with a larger field-of-view for their aerial refueling trainer. He recalled an earlier device for the same purpose which had a 60-degree field-of-view that worked quite well, and so he opposed the request for a more expensive system.

The examples supported the conclusion that in operational environments, processes involving retrieval and comparison of prior cases are far more important in naturalistic decision making than is the application of abstract principles, rules or conscious deliberation between alternatives.

The study identified three primary ways analogy was employed in routine decision making:

1. Understanding situational dynamics. It was evident that some analogues appear to contribute to situation assessment by alerting the decision maker to causal factors operating during an incident. In one case a fireground commander saw a white cloud of smoke and was reminded of a previous incident where toxic smoke showed the same color and density. Because of this he treated the fire as a "hazardous materials" incident by following specified procedures [Klein88a p212-213].
2. Generating options. Analogues were sometimes used to suggest options. The experts used the methods of a case for their problem situation. In this study 14 of 33 analogues were used for this purpose. An example involved an analogue suggesting that the tank strategy of forcing the attackers to use a bounding approach would slow them down.
3. Expectancies. Analogues assisted decision makers in evaluating the probable success or failure of implementing an option. They provided

reassurance for the decision maker that the option worked before and could be relied on. For instance, a tank platoon leader simply used the same avenue of attack as in an earlier exercise because it had worked so well. They were also used to anticipate what might happen if a course of action was implemented. The aspect is essentially "plan repair". In one case a firefighter commander did not use water on hot tar because he recalled an earlier incident when hot tar running off a roof had ignited a secondary fire.

The value of analogues in design tasks was assessed by interviews in which experts were queried as to how they handled design task problems. It was found that informal research and prototypes contributed the most to problem resolution but the second highest category was analogue use.

In the 76 decision points we probed for this study, 13 of them referred to analogues. In six cases the analogues were felt to have been the basis for making the decision, and in the other 7 cases they were cited as being helpful. Thus, the analogues were seen as at least helpful in 20% of the decisions. To put this in perspective, looking at all data sources, 35 were cited as the basis for the decision, 30 as being helpful, 18 as being used but not particularly helpful, and 9 as being useless. So, 13 of the 65 data sources that were useful were analogues. [Klein88a p 213-214]

Klein proposed a descriptive model of decision making called the Recognition-Primed Decision (RPD) model. Its schematic is shown in Figure 2.1 'RPD Model'.

the distinguishing features of the description are that the framework for a decision is the recognition of typicality; options are generated serially rather than concurrently; the first options generated is usually the most promising one; options are evaluated serially rather than concurrently; the attempt is to satisfice rather than optimize; evaluation is through a simulation process of imagining the option being implemented to see if any problems arise; and the decision maker generally is prepared with an action to take rather than having to wait until deliberations are completed. [Klein88a, p 214-216]

Figure 2.1 RPD MODEL

This research was conducted in a behavioral and social science context. Note that the CBR model is very similar to the RPD model. The differences arise from the former's emphasis on implementation on a computer so it functions intelligently and computes efficiently.

2.2 MEMORY AND REMINDING

What are the contents of memory? This clearly is as important as the structure or use of memory. One representation of knowledge is the formalization of our experiences into script-like packages. Scripts are static structures. A script acts as a guide for acting in stereotypical situations. For example, we use a restaurant script which we amalgamated from all of our visits to dining establishments to guide us through a visit to a new restaurant. The script is used to understand actions in circumstances which are expected and to process events along the same lines. In a new restaurant we would still assume the role of the waitress and the purpose of satisfying our hunger. A script eliminates the need to think very deeply, thus mimicking the fact that we do not concentrate our mental efforts on explaining or coping with the obvious. We do not examine the details of waitressing when the server requests our order, we simply say what it is we want. A script facilitates explanation of new events in terms of adapting explanations from similar scripts. For instance, we may be used to ordering, paying and then being served but can quickly adjust to a restaurant in which payment is requested last. Thus a script also supports the notion that most of the time people, when given a choice between thinking hard or adapting new from old, will do the latter.

Rules are another representation of knowledge but for some purposes they are inadequate. They do not capture the reasons or meanings we interpret in events. To add a new rule whenever we learn something would require verification of the rule's consistency. We would have to review the entire body of rules. It seems unlikely that we do this due to the effort which would be involved. A rule base has scattered knowledge and becomes slower as it acquires more rules. This contrasts with people; the more we know, the faster we think. It is likely that much of the time we are not processing rules.

We also have a grouping of knowledge in terms of 'nearness' in the sense that the information we know about a certain subject is readily available due to its connection to the topic. Rule bases are flat with no organization supporting conceptual nearness. Clearly experts do use rules. A rule can provide precise instructions for exceptional circumstances. But, should a precise rule be missing, the fundamental principle which is stripped of the original experiences from which it was generalized is inadequate. When information is kept apart from the instance of its use or the history of its creation, failure is

hard to recover from. Rules can not capture the fact that experts quickly answer with a gut-feeling for the situation as a whole.

How about using scripts or rules for memory structures since they do represent in some fashion, most of the information memory must have? The problem is that memory must foster creativity and be able to change itself as learning takes place. The obvious limitation of script-based knowledge or rules is that they do not support the invention of new ideas. Nor do scripts or rules adequately express the continuous translation of descriptions into labels used to retrieve a memory. Furthermore, sometimes we remember actual experiences and we use the exact recall of the situation. A script is a generalization and a rule is void of experience.

Riesbeck and Schank assert that memory also requires dynamic knowledge structures [Riesbeck89]. There are three types of cases (every experience is a case); ossified cases, paradigmatic cases and stories. Ossified cases are rules. The encoding of a rule is based on the number of repetitions of an experience and the idiosyncrasy of how long it takes a person to perceive and expect constant predictable behavior. That is, as an event occurs more frequently, it becomes a norm and individual occurrences of it become indistinguishable. Thus individual experience and the specific nature of information about the event is dropped as a generalization forms and a standard rule evolves. This is an ossified case or a rule. A proverb is a common form of an ossified case. Rules, like proverbs, are abstractions of unknown origin. Many people share rules while being unaware of the events which generated them. Ossified cases resist change because they are the result of patterns of events that have not changed in many events over a long duration.

If an event does not happen often or is a peculiar variation of a standard event then this idiosyncrasy marks it and prevents it from being assimilated into a general notion. What also often happens is that a case does not have exact replications or it has a lot of replications with small exceptions to its general description. These collective experiences form a paradigmatic case. Paradigmatic cases do not coalesce into rules because they do not relate events that are strictly predictable. They provide approximate predictions of what might occur. They remain in memory as norms to provide 'ballpark' expectations against which other cases can be judged. To reason from paradigmatic cases an understanding of how circumstances differ is used to adapt solutions for a new situation.

Stories are personal. They are unique and full of detail. Children rely on stories because they do not have enough experience to have yet formed a large base of rules or paradigms. The points of a story are buried deep inside its description. A story can relate

to a large variety of circumstances and be extremely productive for cross-contextual application. They are rich sources of creative stimulation .

A uniform representation for all of memory would be inadequate. Representation of rules, paradigmatic cases and stories are required which can express the typical, differential, and creative.

Riesbeck and Schank point out that intelligence need not involve a great deal of understanding but reminding is at the root of how we understand and learn. Reminding reveals the nature and organization of our experiences. It can be used to help us understand memory [Riesbeck89].

For example, a system analyst starts to build the E-R diagram for a purchasing system. He is reminded of a voucher system he recently designed because both involve posting of transactions to a general ledger. His memory needs to have pointers to the characteristics of the voucher system which he will find in common with another type of system such as the purchase order system. A circularity is present: knowing what he will need later, in fact, dictates how he remembers a system. What is important for some given purpose has to be extractable from memory in order to assess what can be of use in similar systems. When the systems analyst completes the purchasing system model it is what is already in memory that affects the incorporation of the new episode.

Reminding is how are we able to retrieve the experience which is most closely related to the new one we are encountering. An examination of reminding is an aid to understanding what role it and memory play in reasoning. Schank defines the following types of reminding [Schank82]:

1. Goal-based reminding. This exploits the notion that when the goal of a situation is recognized we recall a situation with a similar type of goal. For example, if a system analyst realizes that the accounts receivable entity is related to the general ledger entity in several ways but knows that she may only show one relationship in one direction between entities her goal is to represent this multiplicity. She might recall the time she formed an association in order to remove a many to many relationship because in both cases her goal was to describe a conglomeration of concepts in terms of each separate concept.
2. Plan-based reminding. Using plan based reminding facilitates the construction of better plans. We track the plans we create to satisfy our goals. If one knows what actions one will take to cope with a situation, being reminded of a similar plan allows judgement of whether the plan is

adequate or some failure potentially exists. To avoid failure, a plan gets changed before being used. This implies that plans must themselves be memory structures. The plan itself will not be a retrieval key (index) but the implication of the plan (whether it was bad) or whatever circumstances required it are. For example, a plan to avoid getting wet in the rain may involve using an umbrella. If an umbrella is unavailable one could use another plan that solved the problem of an unavailable resource by substitution of a device serving a similar purpose. Thus the plan is remembered by its purpose as opposed to actions.

3. Reminding across multiple contexts. This kind of reminding occurs when a pattern of events, as analyzed in broad, goal-related terms is detected and found to be similar to a previously perceived pattern from another context [Riesbeck89]. For example, sovereignty association can remind someone of marriage breakdown. In the Meech Lake accord the price the rest of Canada must pay to keep Quebec in the Confederation is so high that instead it is better to form some agreement of partial separation. In a marriage where one individual wants to leave, it is better for the other to work out a separation agreement rather than insist on remaining in a situation where too much is sacrificed. This kind of reminding is extremely complex because it uses generalized interpretations of events to find similarities in situations which have explicit and superficially different details. The analysis for the purposes of understanding deals with high level goals, high level interpretations of conditions, interpersonal relationships and outcomes.

4. Reminding through morals. When someone relates a story or anecdote the listener often derives a moral from it or realizes the point of it. This can trigger a memory of the moral or previous episodes for which the same moral applied. The trigger can be a situational or physical similarity. Being reminded of a moral makes it possible to learn greater implications of its meaning. It also implies that higher level structures in memory must exist which correspond to morals.

5. Intentional reminding. This occurs when someone can not find a solution so they force themselves to be reminded of something so an idea will come to them. The action may be conscious or an unconscious thinking device (such as thinking of what would not work). This type of reminding is common and tells us that we have a memory based process which examines our memory indices and tries to use them in addition to a more conventional process that seeks a particular index. For example, we may not understand why a car is not starting. Sometimes we run through incidents where some tool failed to start to see if a reason in those incidents could apply to the car.

The general implication of all forms of reminding is that the structure of memory is crucial to the retrieval of appropriate information. Further, reminding can tell us how we learn and generalize. We learn by understanding something new and then changing what we presently know in order to incorporate the new knowledge. It is important to emphasize that what we term 'good quality' learning involves not just the synthesis of new ideas but their integration with existing knowledge. It is clear that memory must be structured in a manner that it can self-adapt, that is, learn. It must be able to store new memories and to reorganize itself. Its representation must support the generation of expectations.

Children are thought to be more creative than adults. One explanation for this is that they know so little. They take the small stories they have and stretch them to find a hypothetical connection. Sometimes they come up with stunning mismatches and at other times, breathtaking recognitions of similarity. Obviously when you only have a small amount of knowledge to work with, you are forced to postulate or find new relationships between the clusters of what you do know. Because so much is a new experience for children, their former experiences have to be reanalysed and used across contexts. The point is twofold, one: anyone, whether a child or adult, in order to understand something, does so by trying to explain it; and two: one way we manufacture our explanations is by being reminded of what we already have learned and by using the similarities or differences to derive a justification. Children are more creative because while they still rely on being reminded the episodes they retrieve come from a much smaller collection of memories.

The ramification for CBR is that the important aspects of reminding need to be incorporated into how a system uses the case library as memory. An initial assumption is

that reminding basically comprises the retrieval of a closely related experience. But this would be an oversimplification which neglects the realization that reminding and memory representation have a mutual relationship. Reminding relies on memory structures to facilitate fast retrieval. And, in a converse manner, memory relies on reminding to determine its indices and organization. Reminding helps us extract what is significant in an old experience to interpret it so that the new encounter can be handled. The integration of the results of the new experience into memory rely on reminding to indicate what has changed. Memory and reminding are an infinitely processing looped system.

To summarize, reminding implies that a CBR system must maintain a library which can be updated and re-organized by the system, and which uses a storage method which supports easy retrieval. A CBR system needs an indexing scheme which meaningfully marks what characterizes an episode in a way that is distinctive, though not exclusive, and which is suitable as a cue. The issue of similarity implies that matching will often only be partial. Exact matches are not absolutely necessary. The system requires judgement to recognize what constitutes a similarity. Finally, a CBR system must be able to adapt so that information from one case can be transferred to another.

2.3 CBR VERSUS RULE-BASED REASONING

It has previously been stated that there are many ways of reasoning. Because expert systems have been regarded as one of the most reasonable artificial intelligence approaches to modelling general problem solving, a comparison is appropriate. Rule-based expert systems model people by deriving from basic principles the actions which should be taken. Case-based systems rely on reasoning which is closer in style to an expert's thinking. The differences in the rule-based model and the case-based model result in the disparities between the capabilities of each type of system. The basic nature of memory based reasoning indicates that its results will be approximate and quick. Rule-based systems will take a long time to calculate an exact result. As more knowledge is added to a rule base the system becomes slower. This is contrary to what happens with people; as they learn and experience more things they think more quickly. In a rule-based system adding new rules is difficult. This 'brittleness' exists because rules express only superficial theory in a scattered manner. CBR systems are restricted to known situations and produce answers grounded in experience. Rule based systems get their answers from theory but if that theory is complex they can be prone to errors or have gaps in their

capability that are hard to find. The acquisition of the theory becomes a bottleneck in domains where expertise is not well understood or not well expressed by rules.

From the perspective of performance, the short cuts a memory-based system takes result in less computational requirements. Because a CBR system facilitates avoidance of past errors this anticipation also contributes an efficiency saving. A CBR system focuses quickly on the most correct solution available. This too enhances performance capability. A CBR system learns but the learning is not complicated. Rules do not have to be written to be manipulated by meta-rules which redefine the rule base. No causal mode of operation has to be defined or a deep theory of the domain supplied. Experience, in its unstripped detail, is easy to learn from. An experience, since its scope is an entire episode, requires no debugging of interactions.

2.4 AN OVERVIEW OF THE CBR PROCESS

The schematic for a CBR system is shown in Figure 2.2 [Riesbeck89]. There are four phases to the algorithm; input analysis, matching, adaptation and repair.

During analysis the input is examined and indices are assigned. Indices are the cues which will be used when searching the case memory for similar cases and which will later be associated with the input when it is incorporated into memory. Indexing rules are a possible means of the system determining the cues. They tend to be application specific and the rules, which are of a predictive nature, can be learned by the system during its repair phase. Predictive features contribute to the system's anticipatory powers so that the repetition of a mistake will be avoided.

During retrieval the system uses the cues to find cases which are similar (not precisely the same) as the input. To accomplish matching it uses some form of knowledge (match rules in Figure 2.2) to define similarity.

In the next phase, adaptation, the system forms a solution for the input using the best matching case as a basis. Adaptation requires recognition of the differences between the input and case and knowledge of how to deal with them. A library of adaptation rules is one way of expressing this.

If the proposed solution is inadequate the system can try to address the failure by understanding why (via formulating an explanation) and applying causal analysis to generate a repair. The knowledge of how to fix a recognized and understood problem is shown in Figure 2.2 as rules.

Figure 2.2 The CBR Model

The data structures in the system are cases, the case library and libraries of rules for adaptation, matching and repair. In some cases the separation of this knowledge into these logical categories is not paralleled in the system design. For example, the way to repair a case may be within the case itself or matching strategies may be implicit in the CBR system retrieval phase of the algorithm. Rules are actually only one representation that can be used for matching, adaptation and repair. Often heuristics are employed or the behaviour is hard-coded into the algorithm. The processing algorithm is an infinite loop with the outcomes of new experiences being incorporated back into the system as opportunities for future work saving.

2.5 CASE REPRESENTATION

Case representation must meet the requirements of modelling memory. That is, it must support indexing, organizational axes for library content, and permit dynamic re-adjustment and extension of information. Within this framework, the standard techniques to express notions such as structure, abstraction, constraint or inheritance are employed.

Riesbeck and Schank propose MOPS - Memory Organization Packages [Riesbeck89]. A MOP is similar to a script or plan frame except that it is a dynamic memory structure rather than a static knowledge structure. MOPS are also organized into interlinked networks. A MOP describes classes of events in terms of their norms - what the usual goals, events and actors were. A real experience is an Instance MOP. Abstraction MOPS have greater generality in their description and Specializations are more specific versions of a MOP. For instance, your vacation to Cancun is an instance MOP. The MOP which records your expectations about going on a trip is an abstraction and the MOP which further defines the purpose of the trip as a vacation and which contains vacation relevant information is a specialization of the "going on a trip" MOP.

MOPS are joined together by five kinds of links; generalization, scenes, exemplars, indices and expectation failures. The abstraction / specialization links relate MOPS which differ by one level of specialization. Decomposition into sub-events facilitates their being shared amongst MOPS. The scene link acts as the connector between a MOP and its various sub-events. The exemplar links can point to the actual instance MOPs from which a MOP is derived or they can point to prototypical examples of the MOP. An index link is labelled with an attribute-value pair to describe the type of relationship of one MOP has with another in terms of making it distinct from the other. For example, climate - warm would index the going on winter vacation MOP to the

visiting Cancun instance. Finally, the failure link connects MOPS to instance MOPS which involved an expectation failure.

This set of links is complete enough to inter-relate customary episodes, their subtle nuances, good examples and lousy unforeseen outcomes which we remember and learn from. They contribute to memory organization. Scene links create a packaging hierarchy and the index links create a discrimination network. A MOP-based case library changes when new instances are added, new abstractions formed or new indices are assigned.

PARADYME [Kolodner88b] uses a variation of MOPs. It has a hierarchical organization of knowledge and cases and one of its foci is the representational support for parallel case retrieval. Its domain is meal planning (like its predecessor JULIA [Kolodner87]). In PARADYME, the details of any particular event are distributed throughout memory in two ways. First, they are organized into an abstraction hierarchy associated with the kind of situation the event is an instance of. For example, a Mexican meal has its description distributed in Meal, main course, spices to be expected and beverages of choice. Second, a packaging hierarchy associates the details of events with the scenes of the events. For example, meals have scenes which describe the meal preparation, eating the appetizer, eating the main course etc.

Often CBR systems do not bother with hierarchical case library representation and simply use a flat style instead. A factor in this design decision is what information a case encapsulates and the relevancy of the relationships which exist amongst cases. For example, military operation case libraries record historical battle engagements without linking them. No reasonable definition of relationships is possible because "these problems are related to idiosyncratic behavioral reactions of a large number of unpredictable human beings" and "neither the inputs to, nor the progress of, any battle are ever exactly the same as previous combat examples, even in instances in which there is at least superficial similarity". [Dupuy88 p125]

An issue related to case representation concerns whether an entire episode should be described by one case or decomposed so each of its parts is a case. Large cases offer almost complete solutions but it is difficult to use the pieces of a large case because there is no explicit information about how the context of the entire episode affects them. If the theory is available to partition an episode into small cases then they can be stored with pointer links to reconstruct the whole. The portions are easier to access in solving parts of new problems and the applicability of a match can be judged on a part by part basis. The only problem is that additional processes are required to reconstruct it before using the full case.

For example, the distributed representations in PARADYME 'allow the case-based reasoner to use small chunks of cases in its reasoning rather than having to wade through large cases and to allow generalization across scenes common to several kinds of situations' [Kolodner88b p 236]. However, the implication of this is that the retrieval function must be adjusted so that reconstruction can be done if it is necessary. PARADYME had to employ a 'concept refinement step' in the retrieval algorithm to address the issue.

Case-based planning is the application of case-based reasoning to planning. The CHEF system [Hammond89b] represents a recipe as a plan by using inter-related MOPs. Primitive MOPs describe ingredients, steps for preparing an ingredient, the taste of an ingredient and the type of recipe for which an ingredient is suited. A higher level MOP is linked to specific primitive MOPS and states the preconditions and potential failures of a recipe. The MOPs themselves do not contain instructions concerning how they should be adapted nor do they express any explanation for why the steps might be ordered as they are. Other mechanisms (critics) derive the rationale for actions when CHEF needs to explain a failure. CHEF's structures are supplemented with failure information to incorporate what the system learns.

2.6 STEPS IN THE CBR PROCESS

A CBR system will perform one of 2 broad functions. It will perform interpretation or classification of its input via precedent based justification or it will solve a problem in a planning domain by modifying old cases to suit the problem. Systems of the latter orientation use precedent based justification in the stage where they evaluate potential solutions.

For an overview, the process of CBR is as follows. The input is first presented to the CBR system for analysis. The system determines which features of the situation are relevant in the input's description so that it can find a similar situation and start from its solution. It then retrieves, using indices derived from the salient features, all cases which are partially similar to the input case. This is done by establishing a matching between cases. The best matched case is selected for the adaptation phase. In this phase a solution or interpretation for the new case is constructed. Care is taken to account for differences between the two cases and to detect potential failures and thus avoid them. The proposed solution is tested and verified by the system using the case library before being offered to the user. If the user rejects the solution the system tries to explain the reasons for the failure and repairs are made. The solution and any repairs result in an update of memory.

Phase 1: Analysis

Since the input will later be added into the system as a case it is usually referred to as a case - the input case. This may not seem appropriate in systems where the input is not a complete and new problem statement. These systems, such as assistant systems, cope with input which is only a partial problem description though the system may be aware of an overall top-level goal. Since the input is not of the same structure as the cases this introduces a subtle complication in the analysis phase. This will later be discussed in the context of knowledge-based assistance and CMA.

The input case is analyzed to derive an interpretation of what it contains that is relevant to forming a classification or a successful plan. On a trivial level, the obvious characteristics, such as the value of certain attributes are features. More frequently, the relevance is determined by the abstract relationships between superficial features. Common causality or derivation may be important. The results of the analysis are:

1. The identification of features in the input.
2. The assignment of indices for the case should it be added to the case library.

Implicit features can be derived by the system using a checklist prepared in advance of features known to be generally important to the problem. The HYPO system [Ashley88, Rissland88] works in the domain of trade secret law by comparing and contrasting cases. The system has been a priori provided with dimensions as a means of identifying features in the input. 'A dimension is a knowledge structure that identifies a factual feature that links operative facts to known legal approaches to those facts, specifies which are most important for the approach, and specifies how a legal position's strength or weakness can be compared to that of other cases.' [Rissland88] It uses factual predicates to tell whether a feature applies to a case or not. Focal slots emphasize the particular facts which make a case stronger or weaker. From the current fact situation (CFS) the system runs through its library of dimensions to create a Case Analysis Record (CAR) which contains 1) applicable factual predicates; 2) applicable dimensions; 3) near-miss dimensions; 4) potential claims and 5) relevant cases from the case knowledge base.

Another system, JUDGE [Bain85, Bain86], has a similar approach to identifying features. The task of the system is to use a library of previous sentencings for justification to derive sentences for criminal offenders. Given a description of the offence, JUDGE analyzes a new case in terms of:

- escalation of violence
- motivation
- extenuating circumstances
- provocation

Obviously, the analysis factors in both JUDGE and HYPO are strongly related to their respective domains. Furthermore, they have been identified, not by the systems, but for the systems by the designers. While that is a practical approach that works in constricted, uniform domains, it is inadequate in more realistic domains where data is noisy and information ever changing. General techniques which extend analysis to be directed by the system and adaptable to evolution of the system are required.

Retrieval of cases is a massive search problem. Indexing helps by making the retrieval process more selective and reducing the effect of memory size. An index is good if it is distinctive but not unique (otherwise it would not be used as a cue for retrieval). So

the first aspect of this issue is: for its domain, how can a system determine the subtle, extra features which are implicit in the input? Second, it is obviously preferable that the system itself decide upon the indices rather than this being done by its designer. This capability removes the pressure on the designer to anticipate what the system will learn. The system will determine the indices for the knowledge it learns and change its existing indices as knowledge is re-interpreted to have more general or useful meaning.

Several types of approaches in determining appropriate indices have been tried. Inductive techniques use analytic learning methods to identify predictive features. The UNIMEM system [Lebowitz87] used such a weak approach. This approach is knowledge starved as it only uses available evidence not theory and thus it also lacks explanatory justification. It is slow and requires many cases, Some irrelevancies can go into memory and remain there quite a while until they are incrementally weeded out.

Explanation-based techniques can be effective for determining relevant features which are then used as indices. The Barletta and Mark [Barletta88] system worked in a fault discovery domain to provide justification rather than validation or classification. It produced explanations of why a diagnostic action was or was not successful. The system constructed a hypothesis tree to relate a behavioral symptom to potential observables that could have caused it. When the actual facts of the case are applied a diagnostic view was established. This information which stated which action fixed what problem and why it was appropriate was extracted for index use.

Derivational replay involves re-tracing the problem solving, (sometimes by deducing explanations) and using the generalized steps as indices. To further capitalize on generalization there have been attempts to define a vocabulary for describing planning problems, adversarial disputes, and other types of problems so that indices can facilitate reminding across different domains.

The 'indexing problem' is a primary issue in CBR. At the 1989 Case-Based Reasoning Workshop, the editors noted that indexing is one of the open issues that remained to be addressed by the CBR research community [Rissland89]. The important direction to pursue is getting the system to decide on the indices and be flexible enough to generate them dynamically.

Phase 2: Retrieval

Retrieval consists of two steps. First, all cases which partially match the input are found by indexed retrieval. Matching involves forming a correspondence between the features of the input and features of a case. It is approximate to take into account the

uniqueness of each case. Second, the best case, the one which is the most promising in terms of being a suitable solution or precedent is chosen. Some sort of similarity metric has to be applied to make discrimination possible.

Matching implies the capability of determining that two things are similar. The problem is, what does 'being the same' mean? Why is one situation perceived as being similar to another? We are reminded of past events because they have similar goals and plans but what is the similarity amongst the goals or between two plans? The domain seems to have the strongest influence in this judgement. For example, a medical diagnosis system determines two sets of test results as similar if they imply the same illness, whereas a E-R diagram system, would find an E-R diagram of an accounting system similar to one of a purchasing system if they had the same subclass hierarchy implying the same breakdown of function.

Primitive values match exactly if they are equal. They match partially if they share a common abstraction. For instance, a collie matches another collie exactly and a wolfhound matches a collie partially because they are both dogs. These are the simplest forms of matching and are common to most CBR system. In the CYCLOPS [Navichandra88] system the methods are termed direct matching and relaxed matching. Without argument people perceive many similarities in this simple manner.

In more complex situations people apply more powerful matching arrangements. They use analogy. That is, they find common reasons for characteristics being the same. For example, a dead lawn is similar to a seized automobile transmission because the lawn died because it was not given the water it needed and the transmission seized because it was not given the oil it needed. Analogical matching [Carbonell88, Winston80] applies where it is more important to find common causal relationships among attributes in a base and target rather than simply common attributes.

So cases will match exactly if they have exactly the same corresponding features, partially if corresponding features are matched partially, or partially if some corresponding features match partially. The next problem is how to determine which case matches best. To repeat, best means that this particular case provides the best basis for building a new solution. Clearly the contribution of an aspect to influencing the "character" of the episode must be a factor and so must the degree of similarity between a pair of corresponding features.

The most simple similarity metric is to count all similar features and normalize over how many there are. A weighting can be assigned to show the degree of similarity. This approach was successful in the SURVER system [King88] which used expert-assigned

importances for simple weighted assessment. It has also been noted that systems using this method 'perform classification surprisingly well compared to rule bases induced by more complex learning techniques' [Bareiss89b]. The metric, however, does not handle dissimilarities which are relevant and assumes that uniform features exist. That is, every case has a subset of the closed enumerable set of possible features. This approach also ignores the influence of context on each case. For example, the mapping of a car to a truck in one case may be crucial, (because a vehicle is central) but in another case the same car-truck mapping has little to do with the case description. Furthermore, the composition of the library for comparison is influential. If a case is quite unique to the library but later the library acquires cases quite similar to it the method of defining similarity for these cases needs to evolve to be more precise.

In the HYPO system [Rissland88] the approach to weighting is one of least commitment. The system takes three steps to ascertain the relevant similarity of a case: it clusters the factors which are common to the input situation and a case; it interprets what the combined effect of a cluster is with regards to situations being similar; and then it criticizes and tests each of the matching cases to ensure that its particular factors are relevant. A claim lattice is constructed to represent the factors in the input (the root node of the lattice) and the equivalence classes of cases having the same subset of factors in common (descendants of the root node). A case is 'most-on-point' if it has the maximal subset of factors in common with the problem situation. Most-on-point cases receive attention in the second step where it is resolved as to how they can superficially derive conflicting outcomes while sharing features. The most-on-point cases are winnowed in the third step by examining where they lie on boundaries of important factors and whether one contains a factor in its context which far outweighs a cluster in another case. The authors assert that weighting must be deferred until after all three steps are completed. This implies that determination of relative importance is not set down for all future contexts nor carried across problem solving episodes.

A complication is how to factor in aspects which are present in the input and not in the case or present in the case and not in the input. In HYPO heuristics which handle this do so by ignoring certain aspects at the initial stage of retrieval. The authors state that this is acceptable because matching and similarity are motivated by a search for one case anyway [Rissland88]. The two simplifying heuristics which are employed are:

- Temporarily ignore the fact that the most-on-point cases, associated with a particular combination of factors, may differ among themselves as to other factors that they do not share with the problem situation.
- Temporarily ignore difference in magnitudes of the shared factors amongst the most-on-point cases and the problem situation. [Rissland88]

This type of heuristics can be advantageous but adds to the ad-hoc nature of a system because the heuristics are "hand-tuned". A better approach would be to have a complete set of heuristics available to the system and have the ones which are useful in a particular domain be recognized as valuable by the system as it gains experience.

The PROTOS system [Bareiss89a] assesses the similarity of a new situation to a recalled case by explaining how the corresponding features provided equivalent evidence for a classification. Unmatched case features are explained in terms of their relevance to the case's classification and their importance is worked into an overall assessment of similarity based on heuristic evaluation of explanation quality. This approach is not ad-hoc but it does require a body of background knowledge which in some domains may be hard to assemble.

The key open issue in similarity metrics is the same as that of matching; a measure for similarity has to be generated by the system and similarity must be recognizable so that the system can dynamically adjust the measure it uses.

Case Based Inferences

Facts or conclusions derived by rules form the knowledge base on which inductive and deductive inferences work. Cases form the dynamic knowledge base on which case-based inference works. It is appropriate to consider the processes of CB inference at this point - when the best matched case has now been retrieved and the CBR system must use it to generate its solution. Kolodner describes the four case-based inference processes:

1. Transfer the solution that achieved the goal in the previous case.
2. Transfer the solution that achieved the goal and modify it based on differences between the current and previous case.
3. Transfer the inference method by which the previous goal was achieved.

4. Create an abstraction of the problem description from the two cases, extend it to fit the solution to the previous case, and apply it to the new case to create a solution. [Kolodner88a]

Which process is the best one to use depends on a number of considerations. Is there theory or information available to explain differences? Can context be accountable and is it possible to understand interactions? Is the method of inference explicit or derivable? Is the information in the case and its similarity to the input sufficiently rich enough to build abstractions that will support a valid application to the input problem? The domain, representational decisions and similarity assessment are all factors in what type of case-based inference can be made.

Which method is the suitable one to use? When a solution is only a suggestion or when a goal can be achieved by instantiation then a simple transfer (1) will work. It will also save the most time because any long reasoning chain which has been followed in the case need not be repeated. Method 2 is termed comparison-based reasoning. Its strength is that it can be used when exact parts of the previous case can not be used because the entire case expresses the integration of solving simultaneous goals. The differences are the source of information which will indicate what transfer is applicable. Method 3 is also termed derivational analogy [Carbonell88]. It is effective when the same conditions have to be taken into account but the actual circumstances differ, hence, a transfer of the method of making a decision or using the same inference rules is called for. The final method is the only way problem solving short cuts can be exploited. A schema is used to describe both problem statements and then extended to express the case solution. The resulting schema is then applied to the input problem. The schema of the case is an immediate focus with warnings of relevant failures. It helps to weed out irrelevant possibilities.

Phase 3: Adaptation

The best matched case is used in the third phase, adaptation, as a basis for construction of a solution or interpretation for the new case. Cases are also used by the reasoner to anticipate potential failures and modify its solution to avoid them. This is justified by the fundamental assumption in CBR; when two situations are known to be similar and their parts are placed in correspondence, there are underlying causal events, constraints, and properties which lend credence to the belief that other such factors may also be in common. Two steps are involved in adaptation. First, the differences between

the cases are determined. Next, the solution is modified to take these mismatches into account. Any combination of the four case-based inferences may be applied.

As adaptation strategies, these inferences can also be classified as those which operate on structural knowledge and, those that are derivational in nature [Riesbeck89]. The structural approach is to operate on the old solution to yield a new one. Kolodner's first two case-based inference methods based on solution transfer are in this category.

The PLEXUS system [Alterman88] can detect four types of situational differences which exist in planning:

- failing precondition
- failing outcome
- different goals
- steps which are out of order.

When a difference is detected, PLEXUS applies an abstraction and respecialization technique which is structural in nature to resolve it. The process of this technique is to derive an abstraction of the failure and then use another specialization of it as a solution. For example, when a subplan to buy a ticket from a machine fails, PLEXUS generalizes the subplan to one of simply buying a ticket. It then draws upon an example of buying a ticket for the theater at a booth to re-plan to buy a ticket from a booth.

CHEF [Hammond89b] uses critic-based adaptation. A retrieved recipe calling for beef and broccoli will have substitutions of chicken and green beans made and all of the preparation steps will be copied over. The system then uses an ingredient critic to check over the copied steps to see if there are any which are unnecessary, need re-ordering or require insertion due to the substitution of its particular ingredient.

As stated before, analogical reasoning is only possible with derivational adaptation. The methods are transferred as opposed to applying more specific ad-hoc domain dependent transformations. Thus derivational adaptation is also advantageous for cross contextual use of experience. Problem solving knowledge from other domains can be used. The MEDIATOR system [Simpson85] does precisely this. The system tries to derive settlements for disputes. The system contained a plan for settling a dispute amongst children for an orange which recorded failure for the suggestion of dividing the orange in half but which was successful when each child was given the piece of the orange they wanted (peel and segments). Because a dispute between Israel and Egypt was

similar - contention over a resource, the system generated a resolution which gave Israel military control, but retained for Egypt political and economic control.

Concerning adaptation: 'The rules needed for this process are complex and hard to characterize in a general way' [Riesbeck89 p 33]. This is because they are generally ad-hoc and have to be supplied by experts. A primary concern of CBR research is to develop 'general formulations of adaptation heuristics used to modify previous cases or their solutions to fit the new case' [Rissland89 p 2]. One approach that has been tried is to use causal knowledge to formulate general adaptation strategies ([Koton88], in the CASEY system). This however, requires some explicit background theory. Alternatively, focus strategies can be inductively generalized by the system as it watches an expert work [Constant90, Matwin89]. For example, a system could realize that the first difference which should be resolved is the one which presents the most difficulty versus the one which is the most important. The advantages of this approach are that it does not require as large an amount of outside background knowledge (being more heuristic and subjective in nature) and it can be dynamically generated by evidence accumulated by the system as it operates rather than relying on a priori examination of the application.

Adaptation is completed when the proposed solution is tested by the system performing simulation or using case memory to verify that there are no known problems. Memory is annotated with failure information which can be used for this task.

Phase 4: Repair and Learning

Repair takes place after the system has supplied its solution and an outside body has evaluated it. Whereas adaptation fits an old solution to a new case, repair starts with a solution, a failure report and perhaps an explanation and then prescribes the modifications to remove the failure. Repair is at the base of failure driven learning because it modifies memory to express the lesson learned from the failure. Memory is changed by making the important aspects of a failure indices or features of cases.

In CHEF [Riesbeck89] the system compiles the steps and states which combine to cause a failure. The vocabulary of this explanation, expressed in general causal terms, is used to index equally general repair strategies. The repair strategies are indexed under goal interaction structures called Thematic Organization Packages (TOPs). Each TOP is indexed by a particular type of planning problem and organizes a set of strategies for dealing with that type of problem. An example of a general problem description is one stating that plans are being pursued concurrently and the side effect of one violates a precondition of another. Another states that the desired effect of a step interferes with the

satisfaction conditions of a later step. The strategies are general repair rules appropriate for dealing with that type of problem. For example, they might advocate to reorder the steps of a plan or describe further actions that can be taken to recover from a step's side effect. In CHEF the explanation is also used to pinpoint the features which interacted to cause failures and which thus should be looked out for. Learning takes place by the system updating its memory with these warnings.

2.7 SUMMARY

Table 2.1 'Comparison of CBR Systems' characterizes four 'classic' systems in the literature [Ashley88, Barletta88, Hammond89b, Kolodner88b, Kolodner87] and the implementation used for illustration in this thesis, Conceptual Modelling Assistant (CMA). It considers the following aspects of CBR:

1. Quality of Expert's Help: To what extent an expert in the domain is able to articulate the crucial aspects of it to a system designer. If the domain exhibits a lot of idiosyncratic freedom, unstructured methodology or unconstrained choices an expert's ability to assist by providing explanations may be limited.
2. Format of Input: Whether the input data is as complete as the information that is in the system's cases (complete) or whether it is incomplete with respect to the contents of a case (incomplete).
3. Input Analysis: Whether the system has been a priori supplied by an expert (expert supplied) with an analysis scheme or whether the system has some role in deriving how to analyse the input (system derived).
4. Feature Abstraction: Whether features are surface level (shallow) or derived from surface level (deep).
5. Feature Control: Whether the system has been a priori supplied by an expert (expert supplied) with knowledge of what features are important for matching or whether the system has some role in deriving what the important matching features are (system derived).
6. Match Quality Metric: Whether the system has been a priori supplied by an expert (expert supplied) with a metric which pinpoints the best match or whether the system has some role in deriving the metric (system derived).
7. Context of Quality Metric: Whether the same metric is applied to all matches (uniform) or whether information in the match influences the metric (flexible).

	CHEF	JULIA / PARADYME	HYPO	BARLETTA CMA & MARK	
1. Quality of Expert's Help	Excellent	Excellent	Excellent	Excellent	Limited
2. Input Format	Complete	Complete	Complete	Complete	Incomplete
3. Input Analysis	Expert Supplied	Expert Supplied	Expert Supplied	Expert Supplied	System Derived
4. Feature Abstraction	Shallow	Shallow	Shallow	Deep	Deep
5. Feature Recognition	Expert Supplied	Expert Supplied	Expert Supplied	System Derived	System Derived
6. Match Quality Metric	Expert Supplied	Expert Supplied	System Derived	Unknown	System Derived
7. Match Quality Context	Uniform for all cases	Uniform for all cases	Dependent on cases	Unknown	Dependent on cases
8. Represent'n of New Knowledge	Use existing frames	Use existing frames	Use existing frames	Unknown	Flexible heuristic interpretation
9. Acquisition of New Knowledge	Failure Driven	Through new cases	Through new cases	Unknown	Using a training phase

Table 2.1 Comparison of CBR Systems

8. Representation of New Knowledge: Whether the system adds new knowledge in the format of its existing, specified framework or whether the system can flexibly adjust its interpretation of the data.
9. Acquisition of New Knowledge: Whether the system learns solely by using its new input to add cases to its library, or it also relies on failures to motivate its acquisition of new knowledge, or it uses a training phase to learn new knowledge.

In Chapter 3 the issues of case representation are discussed. As the classic systems acquire new knowledge they represent it using the same framework which describes their existing knowledge (see aspect #8). We explain how this strategy limits the potential learning capabilities of a system. The disadvantages motivate an alternative strategy which is later presented in Chapter 3; that is, to use graphs to represent cases. We will discuss the advantages a graph representation, aspects of its implementation, and give examples which demonstrate why graphs are suitable in a class of domains we term 'design task' domains.

Chapter 4 starts by describing how the requirements of a knowledge-based assistant can motivate extensions to CBR which enhance its suitability as an underlying reasoning model. The domains which are best served by KBA systems, design task domains, have some particular problems. In particular, they use cases which are captured at completion not during the course of work (see aspect #2). This means their input (if a system is expected to offer suggestions while users attempt to complete their tasks) will not be a complete case. This further implies that the traditional approach of telling a system how to evaluate its input will not work because it is too difficult to anticipate and stipulate how to handle input at any stage.

Design task domains are also domains where the amount of expertise that can be solicited from an expert is small. This can be due to many factors, as discussed in aspect #1. In many respects the onus is on the system to make decisions which influence and improve its performance. A KBA system must formulate knowledge that can not always be supplied by an expert. Table 2.1 'Comparison of CBR Systems' exposes this general weakness of classic CBR implementations. More often they rely on knowledge obtained from an expert rather than being able to derive useful knowledge themselves.

In general, more decisions with regard to analysis, retrieval, adaptation, and repair have to be formulated and directed by the CBR system itself, not dictated a priori by

system designers and domain experts. The approaches must be general in the sense that they will work in more than one type of domain and be robust enough to deal with the details of multiple domains. In particular, a system that learns what is relevant in its input, what is the best way of matching, what similarity metric is applicable and what transfer of knowledge between a case and the input is profitable is desirable. We describe an extension to CBR which attempts to satisfy these issues in Chapter 5.

3. CASE REPRESENTATION

This chapter begins with an examination of the issues in representing cases. At present the popular medium for case representation is frames. While in general this supports the concept of modelling dynamic memory, a frame's imposed structure is too restrictive for a large number of application domains. This is because in many domains, cases are snapshots taken at completion. It is extremely easy to acquire knowledge in this form but the process by which the case was completed is lost. This implies that the CBR system must derive the important factors in the process so that it can provide plausible responses. The case needs to be regarded as an intermediate source of knowledge on how the task was completed. One goal of a CBR system is to determine by itself, not relying solely upon the expert, what supplemental knowledge should be formulated. Any representation which enforces a static view or interpretation of a case experience, like a frame, may confine the type of knowledge derived from a case and should be avoided.

We then describe the CBR issues and cognitive characteristics of the class of domains which we refer to and have termed 'design tasks'. We explain why design domains are more challenging than the 'classification' domains that have been handled by current CBR systems. This motivates our use of graphs to represent cases. A graph representation for cases is preferable for design assistance, because it facilitates flexible information interpretations of completed tasks and supports different views of the data as information. Furthermore, graphs support domain-independent mechanisms which are used by a CBR system to learn, by itself, what theory is embodied in a case.

3.1 ISSUES IN THE REPRESENTATION OF CASES

'Broadly speaking, a representation is a vocabulary of symbols together with some conventions for arranging them. A good representation is one that has the following coupled characteristics:

- It make the important facts explicit
- It suppresses the irrelevant detail;
- It is perspicuous;
- It exposes constraint;
- It can be computed from natural input.[Winston80, p 690] '

Turning attention to representation in CBR there are many types of information that the representation of an individual ³ case should support. The information may be explicit or derivable from the data in a case. In general, a case must be interpretable as an episode.

Since a case is used to understand something by explaining it, motivational information about an episode is valuable. Goals and plans express the rationale for actions and thus either or both may be included in a case. In domains such as menu planning (JULIA [Kolodner87]) or recipe generation (CHEF [Hammond88]) it is easy to state the goals (for example, prepare a Mexican dinner party for 6) or recognize the discrete steps of the plan (step 1 of a recipe for stir-fry involves preparing the vegetables).

Cases can then be deemed similar on the basis of reasons why something was done or how it was done. Preparing a garden party for 6 is similar to a dinner party for 6. The preparation of vegetables for stir-fry is similar to the preparation of meat for hot-pot.

Goals and plans are also useful for expectation generation. The system can predict what the effect of a certain goal in a specific circumstance will be. A consequence of planning for 6 is that 1 potato must be provided for each guest. A plan provides the steps which may be taken next. The CHEF system expects that after the vegetables are ready it is time to add the seasoning.

One difficult aspect of representing goals and plans in cases is that, in CBR, we do not want to rely on any detailed theory of goal recognition or plan recognition. For some domains, as demonstrated by the previous examples, the theory is superficial and easy to provide. But, if an episode has no inexpensive way of goal or plan recognition using goals and plans in cases becomes counter-productive.

To compound the problem there are domains in which no substantive and unambiguous theory exists and domains where there does not exist any goals or plans which can be usefully abstracted and formalized. For example, starting from a completed and debugged computer program and trying to obtain its functional goals requires a wide range of world knowledge about what its purpose might be. If we need to obtain a design methodology for building the same program, it is questionable whether it is useful to map the program development steps back to a design methodology which may not actually have been followed. It may be misleading to use this derivation as well. Representing an episode in terms of its goals and plans simply does not work in many domains. If they

³We use 'individual' to emphasize reference to the case itself, not the library as a collection of cases. This is implied unless otherwise stated from this point on.

are used despite obvious factors counter to their productiveness, then the quality of the cases as sources of information is limited.

An episode, in its entirety, embodies contextual information which is fundamental to understanding interactions and the applicability to situations. Should an episode be partitioned, it is crucial that each portion retain those aspects of the original context that are influential to its factual details. For example, a visit to a restaurant could be broken into subplans, but if the details of the pay-server subplan are contingent upon the disastrous quality of the meal then this influence must in some way be noted within the subplan. In applications where it is difficult to define which data illustrates context or, information is employed but not expressed by the episode, the case representation must support the system recognizing it by a different means.

An episode typically consists of actions that occur in an order which is imposed by constraints and interactions. A vocabulary is required for representing actions and relationships and a frame can act as a structure for expressing sequence or temporal relationships. Some of the traditional Artificial Intelligence techniques adequately do just this. Data (i.e. factual knowledge) is kept in slots. A slot has an attribute which describes the qualities of values which can fill it. For example, the actor slot with attribute 'person-female' can be filled by 'Jane'.

Slots are the most primitive form of knowledge; they are simply data. Frames dictate the ways in which primitive forms of knowledge can be combined to express more complex concepts. They impose a strict way of viewing data. They can be considered a special kind of graph, that is, one which is always observed from a fixed viewpoint. Though they are not confined to one interpretation, in practise, this is the way they are used. Fixing the structure of explicit data in a case may be too much of a presupposition. Couldn't there be additional causal, behavioral and idiosyncratic (and more) interpretations of data?

A better choice is to preserve the details of the episode in a non-confining representation so that nothing important will be presupposed. Then when a motivation for interpreting an aspect of the episode in a particular manner arises, it is essential for the representation to support seeing and formulating that interpretation.

In CBR systems, some of what the system learns is manifested by it creating links between memory structures or changing the information in cases. Thus the choice of making the system solely use pre-defined structures is disadvantageous because it restricts the type of re-organization that can take place for this learning. Further, the cases which actually comprise the memory exert an influence on the capabilities of the system. The

distinctions between cases, that is the factors which influence whether a new episode might be subsumed by another or be kept in its entirety, are subtle and, once again, any predefined representation for these factors is akin to blinders.

For generality CBR systems maintain a separation between their general purpose component (shell) and domain-dependent component. One has to ask how a shell can be general if it uses too rigid a representation scheme. Certain types of information which may be particularly applicable to the reasoner's intended use could be precluded. One other consideration, which is implementation driven, is that a case representation should be uniform yet at the same time support a means of expressing special cases. Uniformity is also required to have the parts of a case be themselves cases. This relieves other mechanisms from having to differentiate between too many different structures. Without foregoing uniformity, the system also needs to be able to represent changes to meaning 'on the fly' and still be able to work with them. This calls for a representation that can change in the sense of reflecting different information and yet require no change to how the representation is handled by the shell mechanisms.

Another restrictive aspect of frames is that they focus on sequence (plans) or motivational (goals) analysis and typically do not contain any way of expressing visual concepts. There are many domains where visualizations of an episode are frequently used by experts to express their rationale or assessment. For instance, in E-R diagram design an expert may comment that the model needs more relationships because one 'chain' is not 'long enough' or one 'cluster' needs to be more 'bushy'.

The point is that rigidly structured representations are limited to be successful only in domains which exhibit static forms of information and predictable learning patterns. This is the 'classic' set of domains that CBR has been tried on, i.e. classification and planning in simple domains. The episodes in the classic domains are static forms in the sense that the framework of their interpretation never needs to change. Throughout the duration of use of the system there is never any need to change which field from one structure should correspond to a specific field of another. For example, in a legal system, though the 'defendant' slot may be matched with either a 'defendant' or 'plaintiff' slot, there will never be a situation where the 'defendant' slot must be related to the 'judgement' slot. Clearly this aspect of fixed comparisons exists in particular domains but beyond the classic domains the rigidity of this approach would not support changes to the manner in which comparisons are made nor could the system by itself make the changes.

Another aspect of the classic domains is that they 'learn' by updating the contents of memory or by refining memory links with which are associated non-changing

meanings. For example, a new recipe has the same framework of steps as old recipes, every legal case has plaintiff, defendant and arguments, and every report of patient data contains blood test and ECG results. In none of these (cooking, legal, or medical) areas does new information come in a radically (i.e. structurally) different form and, furthermore, it is possible to a priori state how the system will learn.

With respect to case library composition, classic domains have clear-cut and readily apparent bases for generalizations and specializations. The distinctions for discrimination between cases (i.e., deciding on whether to subsume an episode in an existing case) are either unimportant, a priori definable, or can be changed by the system without requiring much background knowledge.

The input of classic domains needs to look like a case because often the same (constant) representation is used for the information in both. Also, classic domains support the input problem being stated in terms which can always be processed in the same way into the same form.

Domains which extend beyond the scope of classic domains in terms of their complexity and unstructured sophistication must use a more flexible representation than one which is frame-based. They require a general data representation that supports many informational views. A solution will extend CBR applicability to domains where one episode has so many interpretations that solely one structure is insufficient to express them and where matching between episodes is not always done in the same way.

Flexible case representation will make CBR feasible in domains in which a case is merely the intermediate form of knowledge for the system. That is, CBR can be used in applications where the episodic information which is easiest to acquire is only a completed result but the stages, retraces and corrections of the derivation are the knowledge which is valuable to the system. For example, in either E-R modelling or assisting with programming, it is easy to get finished models or programs but it is important to use the nonformalizable process as a basis for suggesting help. This involves interpretation of a process which is unstructured and sometimes unpredictable.

Flexibility would be advantageous in domains which must handle case retrieval from input which is only a part of a case. If it could support the visual interpretations that are often used to describe the case then it would be even more powerful. Any independence from goals or planning theory accommodates domains where this information is unobtainable or unwieldy.

Apart from these non-classic domains presenting common challenges to CBR, what do they have in common in terms of cognitive work? The broad answer is that they

entail quite sophisticated mental processing. The textbook knowledge of experts has been supplemented with judgement and decision making skills acquired through experience. The applications are more complex in terms of the expertise that is required to achieve top-level performance. This is because, typically, there are many choices available and subtle factors influence the eventual selection of one. The domains demonstrate a sensitivity to the subjectivity and idiosyncrasies introduced during interaction with people. They require flexibility in making judgements and the disambiguation of non-superficial concepts. They are areas where many lessons are learned from the same events and where on going learning must take place if further progress is to be made. They may be complicated by a lack of formalizable methodology and instead entail unstructured (but more human-like) procedures. We term these domains 'design oriented' to reflect that the cognitive effort is expended designing innovative actions which achieve an overall goal in an excellent manner.

A prototypical example of design oriented domains is engineering design. Engineers generate fairly complex, innovative designs of equipment or systems prior to building them. Helping with an engineer's design is an example of a good use for CBR in which the task clearly has the cognitive aspects previously discussed. Engineers design at some level in an unstructured manner. Even though design methodologies exist, they act more as aids which restrict the amount of 'wandering' towards a solution. This is why engineers with experience but long out of school are more valuable than new graduate engineers who can answer as well on tests of textbook knowledge [Ford90]. Without argument, experts use personalized methods and the more difficult their task is, the more they have to diverge from a strict methodology to come up with an innovative solution [Ulrich88]. For example, in modelling, it is possible to 'reverse-engineer' a design to exemplify a method but in most cases the method will not be the way the design was generated nor would it be in all situations best to advise someone else to proceed according to such a strict method.

Completed designs are a form of knowledge about design that is easy and fast to acquire. However, if they are to be used to help others, they must be treated as intermediate source from which to derive knowledge about the design process. Many designs are typically described in visual terms. For example, we draw VLSI layouts, we draw E-R diagrams, and we draw flowcharts to describe programs. A design will be a source of many interpretations which are suitable for different purposes.

3.2 MODELLER AND CMA

MODELLER is used by a system analyst to help with the construction of a specific, high level data model, an Entity-Relationship diagram (E-R diagram). A system analyst is an expert in database technology, and thoroughly knowledgeable about database management systems (DBMS). MODELLER assists the system analyst in building Entity-Relationship diagrams.

The basic components of an E-R diagram are entities - the objects in a system, and relationships - the behavioral interactions in a system which evolve over time. The diagram illustrates entities as rectangles and relationships as diamonds between directed arcs. The cardinality of a relationship refers to the number of objects of one entity in the relationship that can be related to objects of the other entity in the relationship, and vice versa. A minimum and maximum cardinality can be specified. [Tsichritzis82 p 45] Figure 3.1 is an example of an E-R diagram. An E-R diagram can express the existence of multiple relationships between the same two entities and define the nature of any relationship including its cardinalities. Further, the attributes of an entity (i.e. properties that all data instances share) and keys or sub-keys (aspects which make a data instance unique) are included in E-R diagrams [Tsichritzis82].

Figure 3.1 E-R Diagram Example

(diamonds denote relationships and rectangles denote entities)

Building an E-R diagram is the step in the DB design process that expresses semantic structure and the constraints imposed by a real world situation. It stresses complete logical definition of a system rather than the physical representation of the system's information. The data language used to specify an E-R data model must be flexible so it can support the expression of rich semantic structure. It must also be concise and explicit so that the logical definition can be used to derive a corresponding physical data schema for the system.

The command syntax and semantics of MODELLER's interface are an E-R data language. A graphical interface allows the analyst to draw an E-R diagram. MODELLER has fifteen basic commands⁴ which support entity definition, relationship definitions (though two entities are restricted to only one relation in each direction between them), key and sub-key specification, attribute, sub-key attribute, domain and cardinality (the precise enumeration of a relationship) definition. Relationships can be explicitly taxonomic (*'is Role of'*, *'is Natural subclass of'*), explicit functional dependencies (*'is an Extension of'*, *'is an Association of'*, *'is a Characteristic of'*), or simply generic (*'is Related to'*).

One existing component of MODELLER is a rule-based expert system [Tauzovich87] which, in a limited scope, assists with model validation, construction and evaluation. Consistency checking is provided by contradiction rules. These identify pairs of statements in MODELLER's repertoire that contradict each other. Many of these are implicitly defined by the taxonomy: since kinds are mutually exclusive subclasses, an instance of one of them can not, at the same time, be an instance of another of them.[Tauzovich87]

The expert system can complete the model by making one common transformation which is needed before an E-R design can be converted to an actual physical schema. It transforms a many-to-many relationship into a binary association which has a one-to-many relationship with each of its constituents because a many-to-many relationship is usually not supported by DBMSs. It also, on the user's preference, can apply prespecified uniform defaults to unlabeled cardinalities.

The expert system is capable of detecting some potential problems of a semantic nature. It can verify connectivity, i.e. that no entities are orphans. It can also detect traps which are portions of the E-R diagram that match certain patterns. These do not necessarily indicate that a problem exists, but suggest areas worthy of further investigation. A *'fan trap'* is an entity having one-to-many relationships with two other

⁴ MODELLER is under ongoing development.

entities which are not directly related to each other, but perhaps should be. A *transitive relationship trap* is a direct link between two entities that is perhaps redundant and a *connection trap* finds the only link between two entities is via a third entity through an optional relationship, hence for some instances the link that perhaps should exist is missing. MODELLER also uses causal rules to simulate some behaviour implied by the structure of the model to explore potential conflicts.

After it has verified the consistency of a model and made any necessary transformations, MODELLER executes logical to physical model conversion automatically and provides a data dictionary which can be directly used by a specific DBMS. Conceptual MODELLER Assistant (CMA) is the case-based assistant we designed and implemented for MODELLER.

3.3 USING GRAPHS TO REPRESENT CASES

Graphs fulfill the demands of case representation that have been discussed in 3.1. Graphs are a more general form of frames. Using graphs allows exploitation of the large body of algorithms that have already been written and provides a terminology of graph concepts that can be used to characterize aspects of the domain. In particular, graphs are a suitably powerful technique for case-based reasoning in design domains because they provide flexible interpretation.

'Graphs are simple diagrams consisting of points (vertices) and lines (edges). These diagrams or graphs are used extensively to represent the form of a system. Graphs are simple abstractions of reality. In this sense, graphs are diagrammatical models of systems. Because they are models, graphs are useful in enhancing the understanding of complex systems. As a general rule, any system involving binary relationships can be represented in the form of a graph.' [Chachra79 p 4]

In 1980 Winston implemented a system which learned and reasoned by analogy [Winston80]. This system used an *Extensible-relations representation* where situations were represented using relations between pairs of parts. Supplementary descriptions are attached to the relations when elaboration was needed. This was termed an object-oriented representation since parts of situations are represented as nodes that are tied together with relations forming a kind of semantic net. This representation was selected so that analogies could be formed by pairing situation parts, using acts and other relations as evidence. As well, sentences indicating that an object belongs to some class, an object has some property, or that a particular relationship holds between two objects were able to be

represented [Winston80]. We remark that graphs would have been the most general way to represent extensible relations.

By assigning vertices and edges to types of data in a domain and endowing them with attributes, a graph is capable of expressing situational data and supplementary descriptions in a large class of domains. It is also flexible and general enough to meet both representational and implementation concerns. The reasons for this follow:

Just as frames can do, if it is useful to impose a semantic structure on the library, a graph can still be interconnected with other graphs. In fact, different subgraphs of a graph can form different networks. Graphs can be generalizations, paradigms, or instances. In a CBR implementation a graph be given indices to facilitate fast case retrieval. These indices must indicate how the graph should be viewed. This is an implementation consideration to be handled during system design.

In any domain there are semantic mappings for the primitive abstractions of graphs. An E-R diagram can be represented as a graph, and the geometric abstractions of graphs, such as connected components, cycles, paths, cut-points and bridges, are ideal primitives for expressing semantically valid concepts in the modelling domain. A translation that takes place is the user describing a semantic concept in geometric terms. This is illustrated by Figure 3.2 'Graph Supported Knowledge'. It shows that graph characteristics, such as those previously mentioned and others, provide geometric abstractions which are translations of domain semantics. The domain semantics are interpretations (in terms of predicates) of graph theoretical concepts. These semantics are the building blocks (words) which are combined (into sentences) and used in heuristics. The heuristics express how concepts in the domain combine and form suitable approaches in finding solutions. Heuristics can be employed by the case-based learner. Thus the learner at a high level uses heuristics which are expressed at a lower level by graphs.

Figure 3.2 Graph Supported Knowledge

For example, one submodel may have only one relationship which links it to the rest of the model. The relationship, which is represented as an edge, is a bridge in graph terminology and with the severing of this relationship the submodel is a biconnected component. Figure 3.3 illustrates this: the entities *item*, *bill* and *order* are related and form a biconnected component. The relationship which links this group of entities to the rest of the model via the entity *supplier* is a bridge in graph terms.

Figure 3.3 An E-R diagram as a Graph

The CBR algorithm is dependent on the graph representation but independent of the interpretation given to graph properties in the scope of an application. Thus the shell requires no modifications to its representational components and processes if one domain is exchanged for another. It is important to realize the value of this. There are many domains which have episodes that translate well to graphs and nothing has to be changed in the general CBR mechanisms to accommodate different domains. Figure 3.4 'CBR Shell' denotes domain specific elements of an extended CBR system as rectangles with rounded corners. These elements are either preliminary inputs to the shell (shown with a circle) which remains resistant to domain influence or interpretable outputs of the shell.

Figure 3.4 CBR Shell

A graph supports many different views of data. It can be 'rotated', decomposed into subgraphs, looked upon as a network, viewed in terms of its center, diameter, radius or eccentricities, treated as a cost representation or as a definition of a connection scheme. Still many other interpretations exist. Because a graph imposes less structure than a frame-based case representation scheme, it does not restrict an experience to one rigid interpretation and its basic structure supports many interpretations. With graphs, systems can learn by changing the way an episode is stored. The 'perspective' of a graph can be 'rotated' to support the idea changing how its concepts correspond to those in another graph.

Graphs are extremely useful because they provide a means of expressing visual perspective. As mentioned before, a visual interpretation is often used by experts. Furthermore, taking a visual perspective can exclude distractions which would arise from another view. For example, when the topology of an E-R diagram is an aspect of similarity details such as the specific names of entities in a model do not influence the assessment of similarity.

Another advantage of a graph is that it supports decomposition into components which are themselves graphs. (This is really stating that subgraphs can be treated as graphs, and analogously so can connected components or any special decomposition.) Many decompositions can be useful. For example, if classes of edges are defined, then subgraphs of certain edges show specific classes of data. Sequences of edges defining continuous routes within a graph (called chains and circuits) can show the steps of a design construction or plan and associate non-contiguous groups of actions. The obvious rider is that the transformation to graphs must be well chosen. If it is, the components can collectively embody context, constraints and properties.

3.3.1 The Merits of Graphs in CMA

The advantages in using graphs are illustrated by Conceptual MODELLER Assistant (CMA) where directed graphs are used to represent completed E-R diagrams. The system analyst constructs an E-R diagram using a graphic interface. This interface is transcribed by the tool into a list of assertions. These assertions are the input to CMA. Figure 3.5 shows the graphic interface the analyst works with and the corresponding assertion list which CMA accepts as input.

Figure 3.5 MODELLER Example

CMA makes a graph and some of the standard graph decompositions and attributes the first-level features of its cases. We briefly describe the design of this. We denote the classes defined for this by an capitalized first letter. By subclass, inheritance of super-class attributes and behaviour is implied.

A Graph, as per the formal definition, is a set of vertices each of which may have incoming and outgoing edges. To circumvent repeating redundant work, a Graph has connectionData which is processed information on its connectivity, and, applicationData which is for storing data interpreted according to the application. In the case of CMA applicationData is used to relate the graph to its assertions and domains. Domains in MODELLER are possible sets entities or attributes can be from. Each node or the attribute property of an edge can be assigned a domain and for a graph there exists a group of domains.

A Node has a name and two collections, called inEdges and outEdges respectively, for containing the edges for which it is respectively a destination or source. An

EntityNode is a subclass of Node and is supplemented with an instance of class Entity which contains fields associated with an Entity in MODELLER - *name*, *keyAttribute*, *subKeyAttribute*, *dependents* (for associations), *independent*, and *external*. The AssociationEntity is a subclass of EntityNode and it denotes an entity which is an association. As such, it contains the attribute *constituents* which is a collection of all EntityNodes which are the dependency constituents of the association.

Figure 3.6 Graph Data

(broken line denotes type)

(solid line denotes specialization downwards and inheritance from above)

EdgeWay is a subclass of Edge. Each EdgeWay of the graph has a source and destination. A ModellerEdge, which is a subclass of EdgeWay, has attributes 'relation', 'maxCardinality' and 'minCardinality'. See Figure 3.6. A relationship, such as *natural subclass*, *association* or the generic '*is Related to*' is represented as a ModellerEdge

which has attribute '*relation*' corresponding to its type. The edges are directed and either explicit (an assertion about the relationship has been made) or implicit (implied by an assertion about a relationship between the same two entities but in the opposite direction). Maximum and minimum cardinality of a relationship is represented as an attribute of the edge corresponding to the relationship. Since edges are directed it is possible to represent distinct cardinalities for a specific relationship and its inverse.

Figure 3.7 shows the instantiated values of the graph for the E-R diagram and associated assertions in Figure 3.5 'MODELLER Example'. The properties of the node representing the entity 'supplier' are given in detail.

Figure 3.7 MODELLER Example in Graph Representation (partial)

E-R diagrams are often viewed strictly in terms of the taxonomic or dependency submodels of information they relate. Using a graph representation, these submodels can be expressed as subgraphs. For example, the taxonomic submodel of the model is the subgraph of edges which have a relation of type *'is Role of'* or *'is Natural subclass of'* and all vertices which are either the source or destination of these edges. The dependency submodel's subgraph includes edges of type *'is Characteristic of'*, *'is Extension of'*, and *'is Association of'* and their source and destination vertices. In MODELLER the relation which is the most unspecific is *'is Related To'*. This is a less restrictive relationship which does not imply any default cardinality values. Typically the analyst has to specify the exact values of cardinalities for some *'is Related to'* relations. Since this type of relationship might be singled out for attention in a diagram design step, generic submodels have been defined for analysis, matching, or suggestion generation purposes. Figure 3.8 shows a graph with examples of these three submodels. The asterisk (*) and broken line denote vertices and edges in a taxonomic submodel, the dot (•) and bold line denote a dependency submodel and the '@' and normal line denote a generic submodel. With this example it is easy to see that a subgraph defined on a graph can be treated as a graph and thus no special work is involved to handle decomposition.

Figure 3.8 Subgraphs in a Graph

As part of the knowledge acquisition phase of CMA short sessions observing an actual system analyst interacting with a client using MODELLER were conducted. Later, the analysts were asked for approximate reasons as to why a certain step was taken at a certain time or quizzed about how they conducted the design process. Often the analysts described the rationale for a decision or assessment in terms of a visual characteristic of the model. For example, a generalizing simplification might be made because the 'bushiness' of a group of vertices illustrated that some aspect was common at some level of abstraction to all of them. A graph is a means of defining this and other visual notions empirically. In fact, it works well within a system which can tune empirical definitions using positive or negative feedback.

Topology of E-R diagrams is also a basis for similarity. Two diagrams, though they describe different types of information systems may exhibit the same 'tree' of inheritance or the same size 'fan' in a dependency relationship between an association and its constituents. We used a variety of these topological notions as preliminary match factors and the system fine-tuned their importance or details of their definition according to the quality of suggestion the match produced.

CMA has to cope with the aforementioned difficulty of its input not being in the same form as its cases. Recall that a case is a completed model and the input is a model at some intermediate stage of completion. Graphs make the work of finding a piece of a model in a complete model simple. They can be represented as adjacency matrices and permutations can be tried to find accurate isomorphisms. In the worst case, this may require exponential computation time but heuristics such as anchoring about a center or a fixed point of correspondence such as the top of a taxonomy tree or an association can be tried which reduce the computation cost. Since similarity is not intended to be more than approximate, different measures of correspondence are considered. One measure maps edge to edge regardless of the type of relationship. Another maps edges which have the same type of abstraction (taxonomic or dependency) and yet another lets an edge in the case model map to an absent edge in the input model under the assumption that this might be a good aspect from which to generate a construction suggestion. Since the sequence of E-R diagram design steps is not very constrained (i.e. the timing of incoming information influences sequence more than the rules of E-R design or the syntax requirements of MODELLER) it is quite adequate to use extensions of the subgraph which matched the input as a basis for construction suggestions.

3.3.2 Using Features Associated with Graphs

Graph theory 'has application to problems originating from such diverse fields as psychology, engineering, business, sociology, economics, anthropology, linguistics and geography' [Chachra79 p1] 'some facets of graph theory can be useful in formulating and solving these problems' [Chachra79 preface]

We have experimented and found the following graph notions to be useful:

- Connected components, bridges, cut points and cycles.
- Center, radius, diameter and eccentricities
- Subgraphs
- Subgraphs within a graph of certain size that have links between each other.
- Isomorphisms under various matching rules, with some heuristic short cuts.

We define these notions but only briefly. Historically, graph theory has been developed as a mathematical discipline but there now exists ample references aimed at practitioners who wish to use graphs in their applications [Aho83, Beineke78, Chachra79, Wilson79] These (original) sources cover the applications of graph theory to a wide variety of subjects and contain precise definitions and algorithms which can be implemented if so desired.

A graph G is connected if there is a path joining each pair of vertices of G . A graph which is not connected is called disconnected. Clearly, every disconnected graph can be split up into a number of maximal connected subgraphs, and these subgraphs are called (connected) components. [Beineke78] Figure 3.9 shows a graph with three connected components.

Figure 3.9 Connected Components

An articulation point, or **cut-point**, is a node which if removed from a graph along with its incident edges breaks the connected component of the graph into two or more pieces [Aho83]. An isthmus or bridge is an edge whose omission has the same effect [Gondran, 1984 #6]. Figure 3.10 has a cut-point in vertex 4 and two bridges, the edges (1,4) and (4,5).

Figure 3.10 Cut Points

A **chain** is a sequence of edges such that each edge has one common endpoint with its successor in the sequence and one common endpoint with its predecessor in the sequence. A **circuit** is a chain whose endpoints coincide. The length of a chain is the number of edges it includes. Chains and circuits in undirected graphs are analogous to **paths** and **cycles** in a directed graph with the added stipulation that arcs are directed in the same way.

Starting with some vertex of the graph, we number the vertices in the order of a depth-first search, i.e. we walk, first, as far as possible into the graph without forming a cycle, and then we return to the last bifurcation which had been ignored, and so on until we return to the vertex from which we started. The set of vertices thus met forms the first connected component.

If all vertices of the graph have been met, then the graph is connected. Otherwise we restart the procedure just described from some new vertex; hence we produce a second connected component, and so on. This algorithm is essentially credited to Tremaux (1882). The worked out form of this algorithm is by Tarjan (1972). [Gondran84 p 14-15]

The complexity of this algorithm is $O(e)$ where e is the number of edges.[Aho83]. With minor modifications the same algorithm can be used to find the cut points of a graph. A cut-set is obtained. This is a set, A , of vertices from graph G , such that the subgraph of G without the vertices of A ceases to be connected. Cycles can also be detected at the same time.

In CMA we obtain the connected components of a diagram's graph and further decompose them into cycles and smaller connected components using their cut-sets and bridges. Reduction could be pursued even further, recursively, on the connected components obtained. Using size (the number of vertices) as the index, this gives quick retrieval of smaller 'chunks' against which the input model can be compared. (We perform some preliminary decomposition on larger input models and use the heuristics to guide later decomposition strategies.) The cases are decomposed into useful and logical elements. A good set of building blocks is obtained. A preliminary result has found that cycles are important factors for similarity assessment in the domain of E-R diagram design. In particular, the existence and length of a cycle is important. The connected components were small enough to facilitate trying heuristic methods or exponential time complexity isomorphism algorithms for mapping the entities in the connected component of a case to the entities of a connected component from the input. This will be discussed later. Though it was not actually investigated, it seems feasible that the series of decompositions could be used indicate progressions towards the final model and when some of these progressions can be found in the input model the system may be able to predict what the next logical step could entail for suggestion purposes.

The characteristics of connected components, when used as elemental building blocks, can be useful features:

The diameter of a graph is the greatest distance that can be found in the graph. The set of maximum values (*for each vertex, an eccentricity*) ... is in a sense a measure of the closeness of a graph. If these values are rather small, we can imply that any vertex is relatively close to all other vertices. Using this reasoning, the radius of a graph is defined as the minimum of these distances, i.e., the minimax distance of the graph. The center is the point in a connected graph which has the minimal separation. It need not always be a single point. [Chachra79 p 389-391]

We implemented a modified version of Floyd's algorithm [Aho83] which computes shortest paths using an adjacency matrix. An adjacency matrix is a matrix representation of a graph where the rows and columns correspond to vertices and an element at row i , column j is 1 if there exists an edge between the vertices i and j and a 0 otherwise. In CMA there is no concept of cost in the domain so the cost assigned to each edge is 1 if it exists and 0 if it does not. From the resulting cost matrix, the eccentricities, center(s), diameter, and radius can all be obtained. When two connected components are close in size they can be 'anchored' about their respective centers for mapping purposes. That is, the entities represented by the centers are mapped to each other and then either heuristics or brute force approaches may be tried to find an accurate correspondence for the rest of the entities. The radius and diameter measures are available for hypotheses defining similarity in the same manner as was mentioned for cycles earlier. If the case's connected components has more entities at maximum distance from its center than the input model's, it is possible to hypothesize that a good suggestion was one which advocates adding a new entity at that distance with the same type of relation connecting it to the submodel.

In CMA it is extremely useful to define subgraphs in terms of entities involved in a given class of relationships. One class of relationships, taxonomic, used generalization and specialization ('*is a natural subclass of*', '*is a role of*') links and another used dependency links ('*is a characteristic of*', '*is an extension of*', '*is an association of*'). When supplemented with characteristics such as center and diameter a subgraph was a convenient composition for matching and focusing on a particular aspect about which to generate a suggestion. For example, a dependency and taxonomic subgraph existing in

both the case and input models constituted a stronger match than one where the input case had both subgraphs but the case only had one.

In CMA one goal is to anticipate the correct time to suggest linking two smaller separate components of the input model and to suggest the most plausible relationship and entities which should be used. It is relatively simple to modify a documented algorithm to obtain two submodels of desired sizes from a case model with the stipulation that edges exist which connect them. This can be further parameterized to retrieve models with certain types of relationships. Matching and suggestion heuristics use this utility. If during matching the input model was observed to have distinct components, similar components could be looked for amongst the cases. Later, if a case which matched in this respect was a good enough match to be used for suggestion generation, the linking edges which joined the two submodels could be used to suggest entities to link in the input model and what type of relationship to use. During training, if a decomposition proved useful (by being a factor in a suggestion which received positive feedback) it could be saved with the case as a feature and be directly retrieved via an index.

We can exploit the effort expended in matching two components if the probable mappings for the vertices and edges are saved for later use. They can then later be used during the suggestion phase. As will be explained in more detail in Chapter 5, the first pass retrieval extracts cases with components of either the same size, or with one more entity than the 'focus' component(s) of the input model. It is with these cases that more specific qualities of the match are considered and evaluated.

CMA implements an exhaustive algorithmic technique of computing whether one graph is somewhat similar to another. (Two graphs are said to be isomorphic if there exists a one-to-one correspondence between their vertices that preserves adjacency [Chachra79 p 10].) A connected component is represented using an adjacency matrix. For the case component, we compare permutations of the entities in this representation to the matrix of the input model component. Without any heuristic short cuts there are $n!$ permutations of a n -entity component so we limit the use of this exponential time algorithm to a group of at most 5 components. We define the following ways to measure correspondence between an edge which connects a pair of vertices in the case and an edge which connects a pair of vertices in the input:

1. Any edge in the case can match an edge in the input model regardless of the type of relationship it denotes.

2. Any edge in the case can match a possible edge (it does not have to be present) in the input model. This tries to account for the fact that the case may have extra information which we actually need and can exploit for suggestions as to how to complete the input model.
3. An edge in the case must be of the same class (dependency or taxonomic) as an edge in the input model in order to match.
4. An edge in the case must be of the same class (dependency or taxonomic) as an edge in the input model in order to match or the input model has no edge between these two vertices.

The system looks at possible matches using each measure of correspondence and generates suggestions from the respective mappings. The feedback during training is used to find the best correspondence to use in the domain. After training only the most profitable measure will be used for matching. It is important to note that no particular measure is favored from the start by asking an expert which appears to be the likeliest. The system determines which measures are better through trying them and using the expert's simple positive/negative feedback to make the distinction accurate.

We use the following shortcuts to decrease the number of permutations which could possibly be required:

1. The centers serve as 'anchors' for possible permutations. That is, each center in the input is automatically mapped to a center of the case. Only permutations about the center are tried.
2. Acceptance thresholds are expressed in the matching heuristics and if a permutation has an acceptable percentage of correspondence, no further permutations are tried. This is tied into the feedback mechanism so that a matching heuristic can initially insist upon trying to find a perfect isomorphism but if a less perfect one produces a good suggestion the acceptance threshold is relaxed.
3. If sufficient (defined within a heuristic) positive feedback is received a permutation is favored and tried first all the time.

Once again it should be noted that no opinion or advice is required of the domain expert for these shortcuts to be fine tuned. The system is solely responsible for this.

Once again it should be noted that no opinion or advice is required of the domain expert for these shortcuts to be fine tuned. The system is solely responsible for this.

Instead of using exhaustive search for isomorphisms it would be better to find heuristics in the domain which are strong ways of mapping components. This has also been experimented with in CMA. Hypothetical rules of thumb for mapping components are expressed by the expert initially. Then, as the system uses them in its training mode, their performance is monitored and tuned to eventually result in heuristics which are domain specific and yet computationally efficient. If a heuristic mapping ends up producing suggestions for completion which are just as good as a certain exhaustive method of mapping then it can be substituted. For example, one heuristic maps center to center and then simply maps entities at the same distance outwards from the center with one another. It disregards extra entities at each steps. In a domain where exactness does not yield anything informative this heuristic will suffice and in CMA the system itself learns whether this is so.

3.4 SUMMARY

CMA has shown that graphs provide low level concepts which can be combined and used to express high level interpretations of E-R diagrams and high level strategies for E-R diagram design. Some other domains where the way to use graphs for similar purposes is immediately apparent (e.g. programming, networking). In domains where graph application does not seem obvious a graph's primitive level characterizations still can serve as good building blocks with which to generate hypothetical suppositions which can either be borne out through use or discarded when tried unsuccessfully. Although we have not implemented them, we are confident that other concepts such as degenerate graphs (a graph with only vertices), complete graphs (each node is connected to every other node) and biconnectivity are also useful properties that may be employed to characterize similarity or suggest mappings. It is also important to note that constraint hierarchies and their solutions [Borning87, Freeman90] have a graph representation.

It is also important to emphasize how advantageous it is to exploit a representation that has its foundations in mathematics and has been very useful in other applications. So much of the existing work is formally proven, well understood and published in detail in accessible literature. Only implementation with small modifications is required. The implementation has general purpose use in the respect that it is a set of utilities that is needed and useful for many applications and is part of the shell of a system.

are a representation which supports flexible interpretation and which can provide generality in implementation.

4. CBR AND KNOWLEDGE-BASED ASSISTANCE

MODELLER is the Entity-Relationship modelling tool which assists a system analyst in defining a functional description of a client's information system. Its command repertoire has been described in Chapter 3. CMA is a prototype knowledge-based assistant system for MODELLER which has been implemented with an extended model of case-based reasoning. Implementing CMA has been useful to assure us that our design has been influenced by realistic considerations. This chapter starts with a discussion of what help a KBA system for MODELLER can provide and what difficulties it will encounter and have to be equipped to handle.

Stepping back from specific issues in CMA, next we examine the general demands, motivations and goals of knowledge-based assistants. We consider the nature of the tasks a KBA must perform and hence the type of knowledge that it must keep. This leads to examining how a knowledge base can be assembled, what system capabilities its representation must support and how the system can be developed and validated.

There are clear advantages for using CBR as a model for KBA. Cases, which are easy to acquire, can be used as the knowledge base. This knowledge base of 'experience' relieves the system of needing an explicit theory to describe its application. It is also easy to extend and not subject to 'brittleness' like rule bases are. The paradigm supports the natural idea that expertise is the result of accumulated rich experiences and that experts rely heavily on what they remember of old problems to solve new ones.

KBA also motivates an extension of the basic CBR process to handle its particular demands. An expert must be smoothly incorporated into the process and accommodated by it. He must be allowed to express his expertise in his own terminology. His opinion must be acknowledged and accepted without too many demands to justify it. The system must be capable of providing plausible help at any point during the completion of a task and it must provide whatever help is deemed appropriate by the user.

We conclude this chapter by discussing some motivations for the extended model of CBR for KBA that we present in Chapter 5. We want to define a robust model of CBR which efficiently supports the requirements of a KBA. A robust model of CBR can not rely solely on the a priori advice of an expert for formulating its behaviour. The system must be able to take some initiative, perhaps working with an expert, in improving and directing its performance. It must continue to learn after it is primed with its initial

knowledge. The shell component of the system, that is, the part which is independent of the system's application, must be primarily responsible for this because it makes the model general purpose and exploitable by a large scope of applications.

4.1 CMA: AN ASSISTANT FOR MODELLER

CMA is the knowledge-based assistant implemented for MODELLER. MODELLER is described in Chapter 3. In the interests of generality, MODELLER's rule base must avoid expressing knowledge which is specific to the particular information system being designed. It is very difficult to formulate knowledge of these specific systems in rule form. If it can be done, some systems change so often that maintenance of an accurate up-to-date rule base would be extremely difficult. Also, a reliance on knowledge of this nature limits a system's generality. But obviously information on accounting systems or purchasing systems could provide more meaningful interpretations of the diagram. It would facilitate better consistency and completion advice. The problem is that a rule base is inadequate as a representational source for this type of knowledge.

CMA, if its case library is primed with diagrams which are all of a specific type of system (e.g. accounting) would 'learn' or develop its knowledge in a direction influenced by the typical factors of this type of system. This would be desirable if the sole user of the underlying tool was only working with accounting systems. However, it would limit CMA's effectiveness in helping with other types of systems. An alternative which exists is to first train CMA from a more well rounded set of diagrams. Then it will acquire a more general initial set of knowledge. Later CMA could slowly adapt to the influence of a particular application's diagrams.

MODELLER, at present, uses knowledge which expresses the meaning of its commands. We call this knowledge 'tool level' knowledge. Tool level knowledge is the explicitization of meaning of the command repertoire and expresses how to use the tool to its best advantage. This includes short cuts or heuristic techniques which an experienced manipulator of the tool acquires. It covers the superficial implications of relationship types.

The type of expertise which would improve MODELLER's capacity to assist its users, without focusing on a specific type of system, concerns how to build good diagrams. We term this expertise 'E-R diagram (ERD) design theory' with the stipulation that we refer to an unobtainable (as yet) uncoded body of knowledge that a skilled practitioner has developed in the course of solving many problems. This is knowledge

that is beyond textbook knowledge and which has been acquired by experience. ERD design theory is more general than tool knowledge or application knowledge; with ERD design theory an expert is capable of tackling any application using any tool, without it an expert is limited to either a class of applications or to a certain tool and thus is somewhat specialized. ERD design theory is independent of the tool which is being used and is largely independent of the specific type of system being analyzed. ERD design theory in MODELLER needs to express an approach which is not formalizable and highly unconstrained. If an attempt to define a structured theory is made, the result, given the current state of the knowledge about ERD design in general, would be hard to objectively justify and appear somewhat concocted. Experts can give anecdotal explanations for their choices but not theoretical justification.

'What makes a person a good database designer, in addition to being skilled in applying relevant techniques, is a variety of valuable heuristic rules acquired through experience' [Tauzovich87 p 4]. ERD design theory is what CMA will try to rely upon to provide useful suggestions for completing diagrams. However, it is not the intention with CMA to obtain a formal⁵ version of our so-called ERD design theory as either the form for its initial knowledge base or the knowledge it acquires during training and use. Instead, CMA takes a pragmatic approach; it determines concepts which are crucial to a diagram so that a well-matched diagram is suitable as a source of advice from which good diagrams can be built. Before being trained CMA is unfocused. It is primed with a wide set of aspects which might only prove to be plausible concepts. A concept is the result of recognition, by CMA itself, that an aspect is a significant feature for similarity, case indexing or suggestion generation. Concepts are semantic definitions which can be referred to in heuristics which guide the case-based reasoning component of CMA in using its 'memory' of cases effectively to offer suggestions for building a good diagram.

This formulation of a theory, specifically ERD design theory since assistance for MODELLER is a motivation, sets CMA apart from other established knowledge-based systems. One traditional approach to problem solving has been to rely on an abstraction hierarchy of goals and the formulation of plans which can be pursued to fulfill them [Fikes71, Sacerdoti77, Woodroffe88]. A plan conveys what has been accomplished and provides a context and a set of expectations which the system can use to interpret new input. It supplies the constraints of the domain. In the area of intelligent assistance, providing good quality assistance focuses on modelling the user's intentions in terms of

⁵ (i.e. a stated methodology)

goals and understanding the user's actions in terms of being steps in the plan to reach a goal [Carberry89, Wilensky83]. A well known example is POISE system [Carver84] which is intended to assist with agenda management, error detection and correction, and plan completion.

One shortcoming of plans as they are used in traditional planning systems is that they must be derived from first principles. These principles may be difficult to express or unobtainable. Furthermore, plan knowledge is static in the sense that plans are discarded after use without the system improving based on its experience. Excessive backtracking has to be done when planning for single goals. Application specific or ad-hoc heuristics have been used to provide focus for the search problem but they detract from a system's generality.

Plan and goal recognition is an effective approach but one which is confined to domains where the knowledge is formalizable in this respect. In design domains, such as the one CMA works in, the definition of a structured theory would be difficult to attain because it would subjectively interpret actions and as a result appear somewhat contrived. It is much better to refine expressive, useful concepts in the domain for the purpose of using them to supply good advice instead of explicitly building a plan or goal theory.

At present, MODELLER's graphics provide a stimulus for system analyst and client to add new data to a diagram. MODELLER is truly a 'blackboard' in the sense that the system does not actively participate, it simply shows what a diagram looks like. CMA can increase the active nature of MODELLER to make it a stimulus which is involved in the design process. By offering suggestions based on a growing base of ERD design theory, CMA assists the user of MODELLER with the non-syntactic aspects of an approach to the ERD design process. It is a participant in the synergy between the system analyst, client and MODELLER.

In specifically implementing an assistant for MODELLER which uses ERD design theory numerous issues arise. The nature of design knowledge is heuristic, unstructured and subjective in nature. When the system analyst interacts with the tool and a client watches, the client frequently interjects with information he feels might be valuable. Such an environment is exactly what is wanted yet prevents too much strict enforcement of a rigid procedure. Furthermore, MODELLER's commands are very free-wheeling. They do not need to be employed in certain sequences (beyond the very superficial level). There are lots of choices in the ways in which they can be ordered.

System analysts have to be solicited to acquire at least preliminary knowledge of the theory yet they are often unable to explain why they do certain things or give reasons

for their preferences. When they are stopped during construction of a diagram and asked to explicitly state their options and reasons for choosing the next step, they find it impossible not to be application specific, fuzzy or inconsistent. They use terms like 'it looks too short' or 'it just not the right time yet'. They can easily detect necessary actions but they often can not decide whether an optional action was personal style or vagary. They are an essential 'middle-man' between the system and its knowledge engineer but in the process present additional obfuscation.

The uncertainty or lack of finality in any of the expert's opinions indicate that a refinement process has to be used in the acquisition of knowledge for CMA. It will be impossible to acquire the correct and complete knowledge from an expert and then move it into a system. The expert must be involved in the process of building and refining the knowledge base (i.e. training) and in directing the way the behaviour and performance of the system evolves. A system trained in this respect will be robust in applications which require continually changing knowledge because the system never stops learning from experience.

Furthermore, it is clear that while all users of MODELLER must be experts within the system analyst community there exist different styles, all of which are acceptable. The appropriateness of a suggestion may depend on the style of the user and a knowledge-based assistant system must be sensitive in this respect.

E-R diagram design is sufficiently complex that no layman can quickly acquire the feel for its subtleties. This implies that the system analyst must help in any validation of an assistant system for MODELLER, even if they are handicapped by being unable to articulate all justifications for their evaluations.

Rather than trying to rely on the expertise a skilled practitioner uses, it is definitely simpler to build a syntax-based assistant like MODELLER already has. Such a system relies solely on making explicit the information implied by a command repertoire and explicitly stating certain knowledge concerning the best ways to use the underlying tool. The result, however, is not general and for each different tool a new knowledge base must be formulated. Plan and goal approaches involve more work but they express a richer level of semantics. Since there are application independent theories of planning the systems are more general. More importantly, a system which tracks intentions and understands actions can give more meaningful help to its user. The work, once again, requires a complete and explicit theory. This can only be produced from well delineated and well structured domains. In complex, free-wheeling domains such a system can not use rigid interpretations to figure out a user's intentions. Trying to fit a 'weak' theory ,

i.e. one which is unformalizable, unconstrained and couched in subjective judgement, within previously proposed frameworks is inappropriate. A better approach is needed which will service domains with demands and complexities beyond those served by a syntactic or intention-based approach.

A case-based approach to achieving the effects of such a weak theory offers a number of advantages. If you want to find out how to build good diagrams a logical place to start is from a collection of good diagrams. These diagrams, created by experts, can constitute the case library. Experts build better and better diagrams, this implies that old diagrams play a role in improvement and accumulation of design skills. Experts often use diagrams they remember to help solve similar problems which exist in new ones. The matching or reminding nature of the case-based approach facilitates a way of expressing the theory without being forced to express it as formal knowledge. The theory is, instead, expressed as capturing the definition of concepts and defining the importance of heuristics as they are useful for analyzing input, finding similar cases and providing helpful suggestions.

Furthermore, the extensions to the classic case-based model which are motivated by the goal of providing KBA are in a positive direction for increasing the general power of case-based systems. This will be explained in more detail later.

4.2 GENERAL KNOWLEDGE-BASED ASSISTANCE ISSUES

The issues which arise from building a knowledge-based assistant for MODELLER can be viewed as specific instances of the general challenges a knowledge-based assistant must meet to cope with complex, unstructured and unformalizable domains. Here is a list of requirements that a KBA system faces:

- The nature of tasks which a KBA system must perform are: providing interactive help, as a participant in the task, by suggesting valuable steps towards completing the task well, providing validation facilities, and advising potential subjective modifications which would improve the overall final product. The KBA system must help its user to produce a high quality result. A KBA system must learn not only to give its users what they need but also what they happen to want.

- The nature of the knowledge a KBA system needs is often heuristic, fuzzy, subjective, idiosyncratic, and non-formalizable. It can be very specific to the application, tailored to the specific tool the KBA system supplements, or at the level of methodology. The most general type of knowledge is at the level of methodology. The system aims to produce a weak theory that can supply good advice for obtaining high quality results in the domain regardless of the specific details of the task or the command repertoire of the tool.
- The important aspect of knowledge acquisition for a KBA system is that experts are essential to the process yet they can only be relied upon for approximate start-up knowledge and to judge performance of a system so it can refine its knowledge. Experts must be used to set the system up and then train it with minimal demands on their fluency. Experts are the only people who have an awareness and understanding of the subtle nuances and scope of choices which typically exist in such complex domains. They alone can supply accurate judgement of performance which needs to be subjectively appraised. A knowledge engineer would be incapable of acquiring a sufficient level of understanding of the task to circumvent including an expert in the knowledge acquisition process. While experts may be limited in how articulately they can contribute, this is due to the nature of the knowledge the system needs (see both previous points). They can supply the system with initial, 'ball-park' knowledge but must be allowed to express it in their own terminology. Any request of them for formal explanation or justification is too demanding. What they can further contribute is their judgement of the system's performance. This too requires limited ability to articulate. They can like, dislike or be indifferent to a suggestion produced by the system. The KBA system must take advantage of this simply phrased judgement during a training phrase to direct its ability to provide good help.
Including an end-user or expert further ensures that the KBA system accommodates the specific desires of its user.
- The knowledge base of a KBA system must be able to represent the dynamic nature of the system's accumulation of knowledge. The system

continually refines its interpretation of its knowledge and incorporates new knowledge into its knowledge base in order to improve its performance. No assumptions can be made at the initial design of the system about the structure of its input or knowledge. The interpretation will arise from interaction with the expert when the system adapts itself using the feedback of an expert. The knowledge base must be flexible enough to adjust to new knowledge and a changing interpretation of its existing knowledge. The representation, itself, must maintain consistency so that the system can continue to learn without its internal mechanisms being modified to express new interpretations.

4.3 CBR AND KBA

...there are two major reasons why case-based reasoning is an appropriate technology for building real-world expert systems. One is that the case-based approach is more practical for many domains where rules are very difficult to formalize or too large in number. The other is that the case-based approach better matches the thought processes of domain experts and end users, and hence, among other things, supports a simpler, non-programming interface for domain experts. [Riesbeck88]

In general the case-based approach CMA has taken has several advantages over rule-based systems:

- 1) In certain domains, particularly E-R diagram design (and many others, e.g. programming) it is not only easier to provide cases rather than an initial set of rules but in such domains, it may currently be impossible to capture expertise in a useful and relatively stable set of rules. For example, in CMA complete diagrams which form the initial set of cases (a case library) can be directly read in as they are created with the MODELLER tool.
- 2) By using cases the expert is relieved of the need to formulate rules. This is a task which requires articulateness and consistency often beyond the domain expert's competence even in fairly well understood domains, and is difficult to achieve in complex and subjective domains. Instead the expert's expertise

contributes to the system itself identifying critical features for interpreting input, case matching, and suggestion generation.

- 3) A CBR system can use heuristics in its approach to generating suggestions. Thus the expert during training does not even have to provide carefully chosen examples; he only needs to act as an oracle. His judgement of the perceived quality of the results produced by CMA is either positive, negative, or indifferent. He does not have to explain why. The expert's feedback is used to improve the heuristics which guide the system in its case-based processing.

CMA differs in three main respects from documented case-based systems [Hammond89b, Kolodner87].

- 1) Because CMA functions as an assistant suggesting steps in an ongoing design session, its problem description is typically incomplete at the time it must retrieve good matches from its case base of complete descriptions. Heuristics are used so that the partial E-R diagram is interpreted in terms of what progress it represents and what expectations of extensions to the diagram are plausible. This provides a basis to formulate the first level matching requirements. For example, if CMA determines that more relational information needs to be added to the input diagram the case base is searched for diagrams with components of the same number of entities but more relationships. This works likewise for assessments that motivate relational elaboration or interconnection of subgraphs.
- 2) In CBR systems, features provide the crucial link for short cutting inference to directly obtain solutions; thus one issue debated in the CBR research community is the level of feature superficiality traded off with the computational requirements of deriving deeper level features. One widely used way of dealing with this issue is to 'hard-code' the features of a case, that is, to decide upon an unchanging feature set before the system starts. By contrast, in CMA many possible features of the domain are hypothesized before training. The critical features, i.e. those that are crucial to a match or

the applicability of a heuristic, arise during the training phase when either parameters are tightened or strengths adjusted.

- 3) In CBR systems, rules for assessing cases prior to retrieval, for matching cases and for adapting old solutions to new problems, are typically hand-coded. Even the approach to case-based knowledge acquisition described in [Riesbeck88] requires the expert to manually enter all these rules. Riesbeck's knowledge acquisition interface makes this process only less tedious by providing visual programming aids, thus relieving the expert from having to program. By contrast, CMA acquires much of this information solely from the expert's 'oracular' feedback (positive, negative, indifferent) during its training phase.

Using a CBR model to support knowledge-based assistance presents us with an excellent opportunity to extend CBR in unexplored, potentially worthwhile directions. These have been described in detail in Chapter 2. To reiterate, a CBR system is strongest when the system itself can be relied upon to make more of the decisions that influence its performance. A CBR system that does not have to rely on a expert supplying it a priori with a fixed interpretation of the knowledge it will encounter, criteria for how to match cases to input, or methods to use its best matched case is more robust than one that does. This independence from the expert is advantageous in domains where experts can not supply all the information. It is effective for applications where all the possible ranges of input to the system can not be anticipated because the scope of the domain is wide, noisy or changes very quickly. This independence from 'hand-chosen' decisions also contributes to the system exhibiting more initiative and complements the system's goal to continually learn rather than primarily rely on its initial knowledge. This implies that learning will not be solely the addition of more cases to the case library.

There are three specific places where a case-based KBA system can assume more initiative. First, the system can figure out how the incomplete input it receives should be analysed for the purpose of determining which general class the next step towards completion should be in. This task has to be done by the system because an expert simply can not anticipate every point during the course of the task where assistance may be solicited. The timing depends on various things such as the expertise level of the user, the interchange between any people interacting with the tool or even personal factors like fatigue, boredom or the need for a cigarette. Furthermore, while experts can nominally

identify what type of completion is probably appropriate they can not be expected to formulate the strict reasons why and which cover all situations. The system can be used to objectively derive these over a training period.

Second, a system should be capable of deciding what factors in a case and input should be considered as the criteria for matching and what makes a match good or not. The reasons for this are similar to the reasons for system determined input analysis. The expert can only be expected to supply an approximate idea of what makes a case and input similar or distinct. The system needs to be trained over an 'everyday' set of data so it can refine worthwhile matching characteristics into full-fledged meaningful concepts that reflect the purpose of the cases. Experts can not readily express a formula for judging match quality. At best they might be able to rank criteria by their importance. It has been pointed out that experts have to avoid providing any ranking without considering a range of cases, otherwise they are prone to 'missing the forest for the trees' [Riesbeck88].

Third, since a system can not be expected to initially know what the outcome of input analysis and matching will be, it must be able to determine what in the best matched case should be used to generate a valuable suggestion. Deciding what is important for matching a complete case and less complete input still begs the issue of deciding how to use what is supplemental in the completed case. The only sensible way this issue can be tackled is for the expert to initially give approximate guidelines and then provide opinions which the system can use to refine them. The system must relieve the expert of doing anything more than this, otherwise it will always need the expert when it is required to learn more skills.

For generality, CBR systems maintain a separation between their general purpose components (shell or engine) and domain-dependent components. Another consideration when extending the potential of CBR is to ensure that extensions are made primarily to the shell. Clearly, any extension requires an application perspective. This is the only environment that experts are familiar with and its terminology are the only means by which experts can fluently express themselves. Heuristic data is application specific by nature since it arises from accumulated experience working in a particular area. The challenge is to provide a translation for domain knowledge to a set of uniform flexible representations that the case-based 'engine' can use. The domain concepts must have a form of definition that accommodates changing meaning and which is also stable enough for the system to be able to use the new meaning without the shell requiring changes.

5. AN EXTENDED MODEL OF CBR

In Chapter 4 we set up the functional and design requirements of a knowledge-based assistant that will use CBR to its best advantage. We now present an extended model of CBR for knowledge-based assistant systems which:

- 1) supports two general ways of expressing application knowledge in heuristic terms (classes of heuristics) without detracting from system generality.
- 2) is able to direct, via its heuristic application knowledge,
 - how its input should be interpreted (a KBA motivation)
 - the basis for matching and the quality of a match in the case base (a primary CBR concern)
 - which suggestions are used because they are suitable (i.e. plausible) for task completion (a KBA motivation)
- 3) uses a training phase which
 - relies on the expert to judge only whether its response is good (reward), poor (penalty) or indifferent.
 - has its feedback tied into the adjustment of its heuristic knowledge to influence its performance.

This description includes a discussion of how the design suits the system's requirements and how it contributes to the system's performance. CMA, our prototype KBA system for MODELLER is used for illustration. The description is followed by discussions of aspects of the extended model. The ramifications of the model's approach to handling incomplete input is considered. Knowledge acquisition in our model requires less work for experts and makes the system more of a participant in the acquisition process. This is compared to a specification for a knowledge acquisition interface of a CBR shell by Riesbeck [Riesbeck88]. The extensions to the CBR model focus around input analysis, matching, suggestion generation, and a training stage with a feedback loop. We show how this distinguishes our model and gives it more capabilities than those of classic CBR systems. We show how the way concept descriptions are represented in the model supports robustness and generality in the shell. A description of CMA in operation is presented in Appendix A.1 to demonstrate the overall way the extended model works.

5.1 THE EXTENDED MODEL OF CMA

The specific purpose of CMA is to acquire knowledge of how to build good Entity-Relationship diagrams so that it can provide useful suggestions while a diagram is being constructed using MODELLER. In general terms, the purpose of our extended CBR model is to acquire knowledge of a task which relates to how it is done well so that the system can make good recommendations during pursuit of the task that are valuable to completing it with a high quality result. In CMA we term this knowledge design theory. In general we can refer to it as 'theory of expertise'.

The extended model has three extensions:

1. The processing changes made to generate a suggestion.
2. How heuristic concepts are described and used by the system.
3. How the system trains to perform better.

This section proceeds to describe each part of the extension. This is followed by a discussion of the design related aspects of the system.

5.2 GENERATING SUGGESTIONS

Two stages in the CBR model, input analysis and matching, are extended in our model and a new stage, suggestion generation, is added. KBA motivates changes to the input analysis stage because CMA must handle input diagrams which are incomplete with respect to cases. Input analysis has been changed to use knowledge that tells the system how it should evaluate incomplete diagrams to determine what type of first-level case retrieval should be done. Matching is affected because the system must use heuristic domain knowledge to assist in correspondence and in evaluating a match so that the best matched case can be determined. In order for the system to function as a KBA system it must subsequently use the best matched case to generate some plausible suggestions for the user. The new stage, termed suggestion generation actually replaces adaptation which is not required in this type of system. The new system process for generating a suggestion is shown in Figure 5.1.

Figure 5.1 Generating a Suggestion

5.2.1 The Analysis Stage: During this stage the system tests the input to see if it contains concepts which help it characterize what needs to be done next. It will use this information to decide how to conduct its first-level retrieval of cases. For example, in CMA, a particular part of the input diagram, a cluster, is focused upon for analysis⁶. There are three possible outcomes of analysis:

1. The input diagram needs additional relational elaboration. That is, more relationships or entities have to be added. In this situation, the case library is initially searched for diagrams which have components with one more entity than the input diagram.

2. The input diagram needs additional descriptive elaboration. That is, for the entities and relationships already defined, more detail about them must be specified. Attributes, cardinalities or names may be required. In this situation, the case library is initially searched for diagrams which have components the same size as the focus cluster of the input diagram.

3. It is an appropriate time to link unconnected components of the input diagram together. For example, a taxonomic hierarchy and a dependency path may exist but at present no relationship exists between them though one may be plausible. In this situation the case library is searched for diagrams with two components each of which has approximately the same size as one in the input diagram and which have interconnections.

When the system is not in training mode only the best possible outcome of analysis will be used to drive the case retrieval. The outcomes are judged by a strength measure. This is calculated by testing the input to see how many of the concepts which imply an appraisal are present and how important are the concepts which are present. More will be said later about how these concepts are phrased and how their strength is determined. In training mode the system still tests each set of appraisal concepts against the input but does not use only the best outcome to decide what initial set of cases should be retrieved. Instead a set of cases according to each type of appraisal is retrieved. It will be explained in more detail later how the expert's feedback is used to help the system adjust the appraisal concepts but, basically, when the feedback indicates to the system

⁶ The largest connected component which contains an edge representing the last relational assertion the system analyst has made is chosen or two components unconnected to each other are chosen.

which appraisal was the best it adjusts the concepts it used to influence this to be the more likely outcome.

The left hand third of Figure 5.1 'Generating a Suggestion' indicates the input diagram's focus cluster being selected and tested against the three kinds of appraisal heuristics so the outcome drives first level case retrieval. This processing makes one thing clear: in a CBR system the knowledge of how to appraise the current progress of the task should be phrased so that the system can use it to go looking for the right kind of case.

5.2.2 The Matching Stage: The set of cases which is initially retrieved has been chosen according to the system belief that it is appropriate to recommend a certain type of modification as the next step. For example, in CMA, the set of first-level cases may be retrieved on the basis of the system analyzing the input diagram and finding it likely that, next, relational elaboration is a good thing to do. The subsequent task is matching. Matching is a reminding chore which supplies good diagrams for the system to subsequently use for suggestion purposes. The system's theory of expertise is embodied in matching in terms of how to find a good quality diagram which is similar to the input diagram thus far and how to judge which of many similar diagrams is the best match. In CMA, the match quality heuristic knowledge identifies which diagram properties are significant factors in the assessment of how well the two clusters match. Each matching factor is a separate chunk of knowledge which is also referred to as a match quality heuristic.

The system has a matching driver which, for each set of retrieved cases (three during training and one when not training), pairs the input and a case, (producing a 'candidate pair'), and then queries how each match quality heuristic applies to the match quality of this pair⁷.

A match quality heuristic expresses one concept that should be looked for in the candidate pair. The concept meaning is the same in each of the three possible different circumstances which drove the retrieval but the details which describe it can differ for each one. For example, in CMA, one match quality heuristic may express that a similar number of entities makes the candidate pair match. If the candidate pairing is motivated by relational elaboration analysis, 'similar number of entities' may be defined as equal numbers, plus or minus three. If descriptive elaboration produced the pairing, 'similar number of entities' may be strictly equal numbers of entities. This provides matching

⁷ For clarity, we will refer to this as the candidate pair.

2

of/de

2

MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS
STANDARD REFERENCE MATERIAL 1010a
(ANSI and ISO TEST CHART No. 2)

which depends on the motivation of the retrieval. The intended purpose of the case influences how the input diagram and case are compared and considered similar. CMA does not implicitly presuppose only one reason for using a case which is what, to our knowledge, other CBR systems do. This gives it extended capability to learn better ways to match because it can make finer grained distinctions in comparisons in the context of how the case will be used.

A heuristic is a predicate which uses a concept to express a correspondence. The predicate determines whether a concept: i) is not present in either the input or case or, ii) is present only in case or only in the input or, iii) is present in both the input and case. The result returned to the matching driver expresses the effect of the predicate's finding on the match quality of the candidate pair. This is initially approximated by the expert. Experts are able to indicate simple alternatives like 'it makes the match stronger', 'it makes no difference to the match' or 'it detracts from the match'. In the present implementation, parameter-based heuristics (which will be described later) return a boolean value or 'abstention' to the driver. They have an implicit strength of one. The driver interprets a true result as adding a strength of 1 to the match quality score and a false result as detracting a strength of 1 from the match. An abstention results in the concept being ignored as a factor in the match. An importance-based heuristic directly returns the strength value which should be used for it in the match quality calculation. This value indicates how influential the match factor is on overall similarity judgement of the candidate pair⁸. Thus a non-positive value can decrease the match score and a positive value can increase it. An importance-based heuristic can also choose to abstain by returning an abstention.

The overall strength calculation, or matching metric, that the driver performs is a total of the strengths returned by the heuristics normalized by the number of heuristics which were queried and did not abstain. This calculation is 'fixed' in the system but the values used in the calculation are not. A heuristic itself determines what role it assumes in the calculation. The system has sensitivity to the context of the match and since the system, during training, directs the behavior of a matching heuristic, the definition is under the system's control and not entirely dependent on an expert. The capability is very powerful and unique to CMA. Classic CBR systems 'hard-code' implicit match

⁸ We will later explain more about strength-based heuristics, but it is important to note here that strength indicates what influence the concept has on a match. It does not pertain to the certainty that the concept is a factor and should not be confused with another numeric value sometimes attached to knowledge in rule-based systems termed certainty factors.

strategies. For example, in JUDGE [Bain86] when evaluating a match there is an implicit heuristic to ignore factors that are present in input but not in the case. The system has no way of changing this strategy if it finds a scenario where it is inappropriate. We think that while in other systems it is possible to encode heuristics, these systems rely on their heuristics being specified and finalized by an expert prior to operation, not being changed according to the system's experience, as they are in CMA.

Sometimes when a match factor is not present in the case but it exists in the input diagram (or vice-versa) this is highly indicative of a poor match. Other times this fact can be ignored. There will be occasions when a factor being present in both the input and case is absolutely essential to a good match. The system must pinpoint these circumstances. It does so by tuning the importance of heuristics. For example, one heuristic in CMA searches for exactly corresponding dependency structures in the case and input cluster. If such a correspondence exists, the candidate pair is excellently matched and the case will yield high quality ideas for suggestions. When the correspondence is not present the match is useless. CMA can decide this type of importance because it has a training phase which provides it with a way of changing the strengths of match factors according to context.

One reason that CBR is a good choice for design tasks is that often when an expert is designing his approach is to remember designs similar to the present one. In a CBR system this places great responsibility on matching. The system needs knowledge of what constitutes similarity in the domain and it needs to know how to judge the overall quality of similarity between a case and the input so it can determine the best match. A summary of why the design of CMA's matching meets this requirement in a highly powerful manner follows:

- 1) The extended CBR model's matching recognizes that different reasons for using a case exist and the context of the purpose of retrieval is taken into account during matching. The extended model supports different elaborations of the same meaning depending on the context of use and can adjust the importance of heuristics to account for context.
- 2) In our extended model, so that a factor can affect the match quality differently each time, the matching metric is a calculation parameterized with results from each heuristic. This gives a heuristic more flexibility in how it accounts for context. A heuristic can not only indicate that the correspondence it

expresses either contributes, detracts or does not affect the quality of a match but it can stipulate to what degree it is influential in a match by returning a numeric result or an indication that has an affect on the calculation.

- 3) The system itself can end up finding the make-or-break matching factors in fine grained contexts because it uses information supplied by an expert during a training phase to adjust the behaviour of heuristics.
- 4) The system is flexible because it supports explicit representation of heuristics with the options mentioned in (2) and furthermore it is responsible itself for directing the behavioral improvements of heuristics for their use in matching.
- 5) The system uses heuristic knowledge from the domain to find finely tuned ways of matching. This knowledge is defined in a manner that permits flexible expressions couched in an expert's terms (see 1, 2, 3 previously). The knowledge is a specialization of the general analogy driven matching knowledge a case-based system's shell can be given initially. The matching strategies are semantically based rather than syntactic in origin. They are general purpose advice on matching, not advice on how to match in a particular domain. They provide general principles that can then be specialized to obtain advice on how to match in a particular domain. In CMA knowledge of how to find correspondences in a domain is acquired in a representation that the shell component can use. Because it uses a parameterized calculation the matching driver only depends on knowing whether a heuristic is parameter-based or importance-based to interpret its effect on the match quality.

5.2.3 The Suggestion Stage: In CMA, the system must know how to identify the characteristics in completed diagrams which makes them both high quality and useful. It can then advocate steps which add some of these characteristics to the input diagram. In general, the valuable aspects of a task have to be recognized and then the system can use them as the basis for possible suggestions. In our extended model this knowledge is phrased as suggestion heuristics. The experts initially supply the system with approximate, plausible ways of formulating good suggestions based on an aspect of case

and input. They are further permitted to focus the suggestions around the result of analysis. For example, they might say that:

WHEN the input requires relational elaboration, then, for a candidate pair which has similar '*is Related to*' structures the system should

- 1) suggest adding an equivalent relationship for every extra relationship in the case, or
- 2) suggest adding a new entity and linking relationship to the structure for each extra entity and relationship which exists in the '*is Related to*' structure of the case, or
- 3) suggest adding a relationship which makes the structure cyclic.

and

WHEN the input requires descriptive elaboration, a candidate pair which has similar '*is Related to*' structures the system should

- 1) ignore any extra relationships and entities in the case, and for any relationship which has cardinalities specified in the case suggest the same cardinalities for the corresponding relationship in the input.

Each way of formulating a suggestion is encapsulated as one heuristic. Suggestion heuristics are classified according to the type of input analysis which motivates the suggestion. This is the first condition of the 'WHEN' clauses in the examples. The correspondence used as a basis for suggestion, i.e. the second condition of the 'WHEN' clause, can be the same for heuristics in the same class or in different classes. The two examples above would be incorporated into CMA as four suggestion heuristics. In CMA all suggestion heuristics are importance-based. Parameter-based heuristics are not required because there is no need to refine how to generate an assertion. Instead it is only important to know which suggestion heuristics produce better suggestions.

During training the system generates all suggestions (in CMA suggestions are in the form of E-R diagram assertions) which can be derived from the 3* best matched cases for each possible outcome of analysis (i.e. relational or descriptive elaboration or interconnection). It puts the suggestions in a display which permits the expert, acting as a

* This is a value in CMA that produced an appropriate amount of different suggestions. It is simple to change when the shell is used for a different domain.

trainer, to individually select a suggestion and reward or penalize it, or obtain an explanation of how it was derived. When different heuristics yield the same assertion the assertion is only displayed once but explanation and feedback cover all of its derivations. The reason for generating many suggestions will be detailed in the description of the training phase but it is sufficient to state here that this gives the expert a range from which to judge which suggestions are good or bad and gives the system a way of using feedback to discriminate how useful a suggestion heuristic is.

These processing extensions ensure that the system, after being trained, will offer only plausible and useful assistance. This is possible because:

- The suggestions are derived as a result of the system using the analysis it favours most to retrieve its initial set of cases.
- The reason for using a case is taken into account when matching and when the system calculates which case is the best match.
- The system starts from the most influential suggestion heuristic (i.e. the one with the highest strength) for the analysis class of the best matched case and generates suggestions using them until it has a specified number. (This number is determined by interface considerations.)

To summarize, with respect to processing, the model of CBR for KBA, which takes a partial diagram as input and offers a suggestion to complete it, is extended to improve input analysis and matching, and, to add a new phase where suggestions are generated. Each phase which is extended uses a heuristic set of domain knowledge which the system can either flexibly define or derive the importance of on its own during training. This knowledge can express ideas in the expert's terms and its representation is general enough to support its use by the shell component of the system.

5.3 DESCRIBING AND USING HEURISTICS

Two motivations influence how a heuristic is formulated in our extended model. Regardless of the step in the process when the system used domain knowledge, the knowledge has to:

1) be defined in such a manner that the system can refine the definition on its own or with limited help from an expert. The reason arises from the goal of increasing the system's initiative to learn how to use the concept to its advantage and trying to decrease its reliance on an expert. The goal further implies that these definitions should be evolve without requiring changes being made to the system's shell.

2) have an importance which the system can use to assess its weight in the context of its other heuristic knowledge. Knowing the importance of a concept in given context permits the system to discriminate what is the best possible thing to do next, what is the best match, or to rank the ways of making a valuable suggestion.

Based on these requirements CMA has heuristic knowledge of two basic types; a heuristic is either parameter-based or importance-based⁹.

Parameter-based heuristics permit system adjustment of the concept definitions they use as per requirement #1. They have an implicit fixed strength value equal to 1. They are chosen when the expert can not supply a definition of a concept in exact terms but can propose a use for it. Instead they permit a concept to be initially defined by an approximate range of values. Each heuristic has a feedback procedure associated with it. Parameter-based heuristics use feedback to adjust the definitions of the bounds which define the concept they use. The feedback procedure indicates how the definition of the concept should be changed depending on whether the current definition resulted in a good or bad suggestion. This is how the system acquires the definition of a concept during its training. In matching, because match quality heuristics have different detailed definitions of the same concept which depend on the intended use of the match (see 5.2.2) the feedback procedure takes this into account and adjusts only the appropriate detail of the definition.

⁹ Regardless of whether it is used during input analysis, matching or suggestion generation.

For example, the experts may want to state that the input diagram and a case component are similar if they both have a 'bushy' dependency 'fan'. A dependency 'fan' refers to the shape a part of the diagram has when an association entity has dependency relationships with its constituents. This aspect of the experts' knowledge is easily formulated in graph terms (count the outgoing constituent edges from a vertex which is an association) but the experts are less capable of supplying a numerical definition for 'bushy' or another descriptive term. This may be because they are only guessing that this is true or despite certainty of the usefulness of the observation they are unwilling to decide on the exact values which define 'bushy'. If they do not have to give exact values but can supply ballpark estimates this is sufficient to give the system a place to start. In this example, experts may initialize that a 'bushy' dependency fan has 3 constituents. The system can take over from here, only returning to the experts for their opinion of its suggestions. If the system keeps finding 'bushy' fans and suggests bad changes stemming from the case which had the 'bushy' match then it may be that bushiness as it pertains to similarity has to be stricter, for example 5 constituents have to be present in both the input diagram and case before this makes them similar. The system can use its feedback to evolve the definition in this direction.

Importance-based heuristics indicate to the system how important the analysis, correspondence or suggestion strategy they embody is, relative to the context of the situation. The concept they use does not undergo any refinements because it can be presumed that it is correctly stated directly and will not change.¹⁰ For example, in CMA a cycle is either present or not, but the definition of a cycle is not open to question. The feedback method of an importance-based heuristic is intended to increase the strength 'supplied' by a heuristic whenever a suggestion produced in a context is attributable to the heuristic and deemed to be valuable by the expert. The opposite should occur when a bad suggestion is pointed out by the expert; the heuristic's behaviour should be modified so that it returns a smaller strength in future similar circumstances.

For example, the system has one heuristic (the 'relationally complete' heuristic) that suggests that the partial diagram needs descriptive elaboration next when the number of entities in its focus cluster is the same as the number of relationships in its focus

¹⁰ It is possible to suggest that a third type of heuristic, a compound heuristic, is required because its definition may require refinement (a parameter-basis) and it may be necessary to assert its influence to take context into account (strength-basis). This heuristic requires more thought when considering how it should be affected by feedback and what strategy should be followed for its gradual refinement during training. The system would have to train to first acquire a definition because strength should only be fine tuned once the definition is stable.

cluster. The expert is sure that a range of plus or minus 3 is a correct definition of 'same number of entities and relationships'. The heuristic may start with an initial strength of 1 whenever the definition holds for the input, and -1 when it does not. If the concept is often found to be present and as a result the system on these occasions makes a good suggestion, this heuristic should become more important in determining that relational elaboration is necessary. In comparison another heuristic whose concept is found to be present less often when a good suggestion is produced should not have as great an effect on the assessment of a similar input diagram. One feedback scheme we use is a traditional and conservative one. The size of an adjustment (i.e. how much the strength is modified for the context) is smaller the more successful a heuristic has been and is larger when the heuristic has exhibited poor performance. Bad performance requires drastic change and good performance implies that changes away from what has worked quite successfully before should be conservative. The result of training on the example heuristic here is that the system learns how important this particular definition of relational completeness is to deciding whether an input diagram requires relational elaboration.

It can be seen that the definition given by experts is in graph terms to the system. The system keeps the experts' term of 'bushy dependency fan' associated with the graph definition it derives. What arises as a result of training then, is a definition for a semantic term of the domain. As another example, the 'long enough taxonomic path' heuristic tests the length of a taxonomy path against its current definition of what 'long-enough' is. If the path tests negatively but feedback from the oracle indicates that the path was sufficiently long then the definition of 'long-enough taxonomic path' is adjusted.

CMA itself plays an important role in defining a concept. It makes changes based on receiving feedback. Not only does this increase the initiative of the system and decrease its reliance on an expert but advantages similar to those associated with empirical machine learning are present. The values the system derives can be incrementally modified in a consistent and objective manner based upon real experience.

5.4 TRAINING THE SYSTEM

It has been mentioned previously that one goal of CMA's extended CBR model is to provide KBA for domains where it is impossible to acquire expertise directly and exactly from skilled practitioners prior to the system going into operation. In these domains the expert can not be solely relied upon to a priori supply all the necessary knowledge that the system needs or to articulate it clearly. For example, in E-R diagram

design, experts can not necessarily state how to build a good diagram nor can they provide precise definitions for important characteristics of good diagrams. Nor would it be safe in complex domains to believe that the system could be given a set of knowledge that would not require revision as time progresses and the characteristics of the domain change. In the course of being used a long time by one user a system should accustom itself to his or her particular preferences.

This situation has influenced the approach CMA has taken to knowledge acquisition. The decision has been made to use experts for knowledge acquisition in two simple ways. Experts provide the system with an initial set of approximate heuristic knowledge and, during a training phase, they judge the help the system suggests in order to give the system feedback to fine-tune its initial knowledge base. In this latter phase the expert works as an oracle. This is shown in Figure 5.2. This approach should decrease the amount of time experts need to spend in knowledge engineering and decrease the work they have to expend to describe their skill to a layman. The experts still have to think. They must clarify in their own minds the reasons they have for their opinions and ensure that they judge consistently. However, they don't have to spend time trying to construct explanations for the knowledge engineer.

Knowledge acquisition, in the form of the system deriving better definitions for the conceptual knowledge it uses and realizing the importance of concepts in situational contexts, is the ultimate goal of the training phase. During the training phase the system executes every processing step shown in Figure 5.1. The idea behind the training phase is that the system tries all the ways it has to generate a large group of suggestions. It records what knowledge was a factor in deriving each suggestion. Input analysis using appraisal heuristics is executed but, instead of only using the strongest result (relational elaboration, descriptive elaboration or interconnection), first-level retrieval collects an initial set of cases for each of the 3 possible results. The system then applies its match quality heuristics to evaluate every possible candidate pair in each initial set. The best three matches for each analysis result are used for suggestion generation, instead of only the best. All suggestion heuristics for an analysis class are tried on the corresponding three best matches. Redundant suggestions are only displayed once. The feedback loop starts when the experts are allowed to pick which suggestions they think are helpful to building a good diagram and those that are not appropriate. They actually can select a suggestion and reward it (which tells the system it is good) or penalize it (which tells the system it is poor).

Figure 5.2 Using the Expert as An Oracle

The system has recorded for each suggestion, all the suggestion heuristics which generated it, all the cases which motivated it, the match quality heuristic results for each candidate pair and the appraisal heuristics which motivated the first-level retrieval. Each heuristic has a feedback procedure associated with it. The procedure enacts changes to the heuristic definition (if it is parameter-based) or the heuristic's influence (via the strength of an importance-based heuristic) based on receiving a penalty or a reward. This is shown in Figure 5.3.

Different feedback schemes can be implemented. Experts may be able to supply an appropriate gradient in parameter or strength adjustment or the system can use a scheme which determines the size of the change relative to how well the current definition has proven. We have previously suggested a well proven conservative scheme in which rewards contribute to the inertia of a definition. While the finest level of using feedback is at the level of a single heuristic, a 'meta-level' feedback procedure could try to perform similarity-based generalization of heuristics by observing how various heuristics are performing. A more final approach to feedback, which relies completely on the expert, is to prompt the user for what change should be made to the concept a heuristic uses. The system could itself advocate adding a new heuristic to clarify why two heuristics with conflicting meaning keep turning up together or ask if some value should be added to an enumerated range definition. Turning to the expert can be a catch-all for unanticipated contexts and should be used as a last resort.

Two examples of heuristics follow:

1. Use: Appraisal Heuristic Class: Relational Elaboration Type: Importance-based
Description: If a taxonomic subgraph does not exist in the input diagram then this supports relational elaboration as the next step with strength 1.5.
Result for Current Input: True (no taxonomic subgraph exists)
Feedback: Reward (a relational assertion which came from a case retrieved to suggest relational elaboration was deemed good by experts)
Update: Historically this heuristic has been a factor in a good suggestion 4 times out of five so increase its strength to 2.0
2. Use: Match Quality Heuristic Type: Parameter-based
Values: cutOff = 60% for descriptive elaboration, cutOff = 50% for relational elaboration, cutOff = 90% for interconnection

Description: compute the most accurate isomorphism with the focus clusters of the candidate pair, save the resulting mappings, return true if the accuracy of the isomorphism is at least as good as cutOff.

Result for Current Input: True, best accuracy was 60%, **Class:** Descriptive Elaboration

Feedback: Penalty (a cardinality assertion which came from a case retrieved to suggest descriptive elaboration was deemed poor by experts)

Update: The cutOff point must be higher, increase it to 70%

The outcome of the training phase is that the system has acquired a knowledge base with better definitions. This will be instrumental in its being able to generate highly useful and plausible suggestions beyond training. After the training phase, the system is restricted to feedback of a very narrow nature. If the user chooses to use a suggestion the system offers, the system will know that this was a good suggestion. It no longer has a way of being told that it has offered a poor suggestion. The best approach might be to make no more adjustments until strong evidence has been accumulated. Thus the system learns more slowly after training. This is similar to the learning curve people encounter when learning a new skill. They learn a lot rapidly at the outset and then the rate of knowledge accumulation and quantity they learn tapers off.

To summarize the important aspects of the extended model: the system uses heuristic knowledge during input appraisal, matching and suggestion generation. An approximate initial definition of this is supplied by experts. The heuristic knowledge is formulated so the definitions can be manipulated by the system to make them accurate or the system can change the importance of the knowledge to reflect context. Appraisal heuristics analyze the input diagram so the system can decide what the next type of suggestion should be. This influences the initial set of cases it retrieves from the case library. Match quality heuristics express correspondences for candidate pair matches. Suggestion heuristics indicate what suggestion should be made using the case. The system is trained to acquire more accurate definitions of its concepts. During training it tries all possible ways of generating suggestions and it shows these to experts. Experts choose the suggestions they feel are good and reward them. They penalize the ones they feel are inappropriate. The system uses this feedback to improve its knowledge base. It changes bounds or strengths for heuristics that incorrectly affected a suggestion and recognizes that others are accurate. A more detailed description is presented in Appendix A.1.

5.5 CASES AND INPUT

When using CBR for KBA in design domains the cases will be completed designs. The input is an intermediate version of a design which implies that it is an incomplete case. This is one distinction between a KBA system based on CBR and classical CBR systems. In classic CBR systems the input looks like a case. Data may be missing but this fact is known so the system is primed to deal with such situations. The system is not involved in any incremental interactive processing.

KBA presents case representation issues. A case must be decomposable into smaller components that themselves look like cases so that the input diagram can be matched to them. The system must be able to form different interpretations of the case components when it sees that they correspond to a particular type of input. In Chapter 3 we have explained why graphs are an ideal representation for doing this. Graphs are flexible enough to represent knowledge in different domains and robust enough to represent changing meanings.

Cases are not directly useful. They do not indicate to the system how they can be used to help the user because they are only a final result. The theory about what makes a case good must be formulated and this is the knowledge used to assist users. In CMA a case is a good diagram but the diagram does not contain a direct indication of what makes it good or what were the steps taken that made it well constructed. The system needs to build interpretations of a case (heuristic knowledge) which embody this knowledge. Formulating these interpretations around the processing stages of a case-based reasoning model is a pragmatic and effective way of doing this. The system's matching knowledge expresses what concepts are important in a case to make it a good match. Complex domains have *importance-dominated matching*: 'The similarity between two situations is measured by finding the best possible match according to what is important in the situations as exhibited by the situations themselves [Winston80]' and *description-determined similarity*: 'A situation is similar to another if the important relations in their descriptions can be placed in correspondence' [Winston80]. The suggestion generation knowledge can be used to perform *analogy-driven constraint learning*: 'A constraint is learned as a byproduct of mapping the parts of a situation in a well-understood domain into the parts of another situation in an ill-understood domain' and *analogy-driven learning*: 'Some questions about a situation ask if a particular relation holds. Causes found in a remembered situation can supply suggestive precedents' [Winston80].

5.6 KNOWLEDGE ACQUISITION

In our extended model experts are asked to provide loose hypotheses of how input should be analysed, how a case and input should be matched and evaluated, and how the best matched case should be used to generate a good suggestion. They are allowed to do this in the application vocabulary. The only further demand on them is to participate in the training of the system by offering opinions which are strictly of a 'I like', 'I don't like', or 'I don't care' nature on its output. This meets the goal of demanding only minimal requirements for explanation while still using experts' opinions.

The issues arising from knowledge engineering and knowledge acquisition in case-based systems have also been recognized by Riesbeck [Riesbeck88]. He notes that experts can not express how they solve a problem using a programming vocabulary. Though the knowledge engineer is an essential middleman in the knowledge acquisition process the position increases cost and decreases efficiency. Formulating the experts' ideas into the representational language of the system converts the knowledge into a form which the experts find difficult to understand and thus they experience difficulty in validating a system because even though the correct answers may be generated for certain examples, they may be unable to tell if those answers are being derived in a reasonable way. An expert can relate to a case.

Riesbeck describes a case library acquisition interface that was developed as part of a CBR shell. 'A case library is a set of cases plus a set of analysis and matching rules that determine how cases match each other. The interface is designed to make it simple for the domain expert to input new cases, find old cases, modify how field values match, and what fields are most important in ranking the similarity of cases' [Riesbeck88 p312].

Riesbeck defines a functional specification of the interface which he believes is satisfactory for the requirements.

The interface is designed to help the expert enter without programming:

- analysis information, i.e. rules about what to look for in input cases,
- matching information, i.e., rules about how to match cases, and
- adaptation information, i.e., rules about how to make old solutions fit new problems. [Riesbeck88 p 316]

While Riesbeck proposes to supply experts with an interface which is less oriented around programming, the solution still emphasizes work being done by experts. Initially experts must pre-sort the cases into piles of similar cases and exceptional cases. 'The main purpose of the pre-sorting phase is to prevent the domain expert from adding cases on a case-by-case basis, with no feeling for the overall relationships.' [Riesbeck88 p 316] To change the matching criteria, a display is used that presents the most important matching factors first and a highlighting of those which contributed to the match. This gives the experts a clear indication which factors need to be moved up or down to get a different grouping for matching. 'This kind of display supports a very direct manipulation technique for changing the importance of fields. To make a field more or less important, the expert can point to it with a mouse and drag the field into a higher or lower group. The expert does not have to assign explicit "importance numbers" to fields, and it is always clear in what direction the changes in importance will affect this particular match.' [Riesbeck88 p 319].

It is important to note that while the interface is graphically oriented which makes it more conducive to exchanging information with experts, the underlying system still requires knowledge formulated in rule form. A sizable amount of care has to be taken by the experts to ensure that their changes and judgements result in consistent rules for the overall relationships in the domain.

One methodology for the expert building the library is this. The pre-sorted cases are entered into machine-readable form. The expert makes a selection of prototypical examples and adds them as cases into the case library. The knowledge acquisition starts with the expert testing the system. The expert decides which case (the target case) should be retrieved for a prototypical input case he selects. The system is presented with the input case and it displays the case it would retrieve and consider the best match. If a case other than the target case is retrieved, the expert continues to modify the rules until the input case retrieves the target case. 'There are 3 basic kinds of changes the domain expert can make that will affect case retrieval. The expert can:

- make certain field values match better than others,
- change the importance of some fields, so that matching on these fields has more or less effect on the total match,
- add new calculated fields to the basic case form, so that the number of matching fields is increased in certain situations. [Riesbeck88 p 317]

It is important to recognize that in this process the onus is still on the experts rather than the system to change the rules to make the system behave a desired way. The system assumes no initiative in modifying the rules. It is relatively dumb and passive in the knowledge acquisition process. It simply acts as a processing and output device rather than as a participant learning knowledge. A system will probably always need an expert's judgement because it has less background knowledge but it could be used to make objective and efficient changes. This is what CMA does. In CMA the feedback procedures allow the system to change concept definitions in a non-arbitrary manner according to evidence from a real experience.

With respect to entering episodes as cases Riesbeck states:

In form-based domains, ... the obvious interface is to present the forms on the screen to be filled in In a domain that doesn't have standardized forms, it is a good idea to first invent one. Designing forms is a good warm-up exercise for understanding a domain. It forces the domain expert to categorize and organize aspects of the domain in a manner that is consistent across examples. ... We'll assume here that a standardized well-structured form does exist, or can be invented. [Riesbeck88 p 316]

This assumption restricts the domains which can be served by this approach to knowledge engineering. In design oriented domains there may be too many choices, judgements and unstructured processes to be able to define a form. Further, there is a hidden assumption in Riesbeck that the input will be like the cases. The same form will suffice for the cases and the input and the rules will always consider the complete structure of the form with an implication of what information should or should not be present. In domains where assistance is sought during the incremental pursuit of a task this assumption is invalid. The approach to knowledge acquisition has to accommodate the fact that the input is only a part of any case. This influences the knowledge the system needs to analyze the input and makes it different from the knowledge which the system might use to match a case with the input. In CMA this distinction is served. Heuristic knowledge is acquired specifically as it concerns input analysis and then analysis influences the first level type of retrieval. Another type of heuristic knowledge which is

subdivided into knowledge to be used based on the basic type of input appraisal is used for matching and match evaluation.

Our extended CBR model also differs in another aspect of knowledge acquisition. Because it tries to incorporate the system more actively into the knowledge acquisition process, it reduces the effort required by experts. The intent is to reverse the feedback roles. The system should accept feedback from the expert and modify itself rather than the scenario in Riesbeck where the system is tested and supplies feedback to the expert who modifies it. It is easier for the expert to pick the best answer amongst a variety of choices and then to rely on the system to adjust the knowledge appropriately.

It is easy for the expert to give an opinion but both Riesbeck and the extended CMA model face the same problem: how to ensure the expert's judgement does not lead to an inconsistency with the rest of the domain. It can be debated which approach is better. Riesbeck firmly places this responsibility on the expert and makes the expert aware of it:

There is an important caveat to this operation, however, that must be made clear to the domain expert. Changes to the importance of fields should be made very sparingly and for good reasons. Fields should not be pushed around just to make a particular match work, because this can undo work on prior matches. [Riesbeck88 p 320]

When experts are used as an oracle as they are in CMA, giving them a grunt-like response medium, they may feel that they do not have to give well founded opinions. This is misleading; they still have to think. The domains are presumed to have *historical continuity*: 'A situation that is similar to a past situation generally leads to similar results or conclusions' and *constraint-determined importance*: 'The important relations of a situation are the ones explicitly said to be important by some teacher or implicitly known to be important by being involved in constraint relations. Often these constraint relations have to do with various forms of cause' [Winston80]. The experts must clarify in their own minds the reasons they have for their opinions and ensure that they judge consistently. In CMA the experts need to be aware of the premises for their decisions but they are not forced to always fluently express their explanations in the system's terms. This saves them the time they would otherwise spend trying to explain them in concrete terms to the knowledge engineer.

While the Riesbeck reference describes the issues of knowledge acquisition for a CBR shell and proposes how the domain expert interface should function, it does not describe the design and implementation of this functional specification. This has been done, in this thesis, with CMA and its extended model of CBR. Furthermore, the solution is general and robust.

5.7 DISCUSSION OF THE PROCESSING EXTENSIONS IN CMA

It has been noted that:

the expertise that must be given to a case-based reasoner is more than just a collection of cases.... The domain-dependent knowledge that a case-based reasoner must have are:

- the cases themselves
- rules for analyzing the cases, prior to retrieval,
- rules for matching cases, and
- rules for adapting cases to a new situation. [Riesbeck88 p 315]

In this sense, the knowledge structures CMA uses to perform knowledge-based assistance fulfill the same purposes as rules. Heuristics have been grouped according to where changes need to be made in the general processing loop of a CBR model (recall Figure 1.1 and Figure 2.3). A KBA system will train itself so it can perform better at the following points in the CBR process: input analysis, matching, and suggestion generation. In design task domains where completion is the assistant's task there is no need to perform adaptation. One effect is more emphasis at the beginning of the processing loop. The system must distinguish the knowledge needed in the input analysis phase because the input is incomplete with respect to a case.

It has always been recognized that matching is the most important capability a CBR system must have. Our contribution is to let the system acquire this capability instead of totally relying on getting it from an expert. This is supported by the two types of heuristic knowledge which the system uses. Parameter-based heuristics can have concept definitions modified by the system. Importance-based heuristics can have their importance appropriately changed by the system. To our knowledge this has not been done before in any case-based reasoning system.

The extended model also has a unique way of appraising similarity which is highly sensitive to context. We have discussed in Chapter 2 how classic case-based systems, e.g. JUDGE [Bain86] use a set of pre-selected features which are a priori ranked in terms of importance for matching. The systems have one implicit purpose for using a case and this is reflected in which features are selected and how they are ranked. CMA, when matching, does make use of a set of selected concepts but the finalized set arises from training which exploits experiential evidence to draw its contents from a wider hypothetical set. In CMA, the definition or importance of a concept is not predetermined. Again, this contributes to the system's flexibility and robustness. Furthermore CMA is designed so that the purpose of a case is accounted for when deciding what is the definition or importance of a concept. Heuristic knowledge is explicitly classified by purpose which means the system can handle using cases for subtly different reasons and improve its value to its users.

The classic systems can not change their specific approach to similarity assessment themselves. It must be done by hand. They do, however, track their successes and failures and use this information to modify an episode. Annotating a case with potential failures or warnings of failed expectations changes the way a case matches according to the predetermined feature ranking list. In contrast CMA avoids changing its cases. CMA tracks its successes and failures in similarity assessment by changing the heuristic knowledge it uses to interpret them. This approach is better because the correction is not exclusively hidden inside one case. Changing the interpretation instead makes the knowledge more generally available. After all, the features of memories do not change, it is the interpretation of the memory (or of a case) for purpose which really changes when evaluation shows whether the memory was used in a valuable manner.

Another goal in matching is to have the way in which a case is used be relative to the composition of the case library. That is, the way in which a case differs from the other cases or how exemplary it is should drive how its match quality is appraised. Systems which do not acquire their knowledge on their own are brittle in this respect. The knowledge they are given must cover the overall nature of the domain but can not anticipate the detailed aspects in matching which arise from examining the composition of the library. CMA is more flexible in this respect because during training it uses its case base to fine-tune its matching knowledge. The case base can be selected to contain classic prototypes which are representative of the scope of the domain and to include important exceptions to the prototypes.

To simplify their tasks classic systems use implicit heuristics that help them to retrieve less cases on first pass or to scale down the number of features they consider for match quality. For example, in HYPO [Rissland88], to produce the initial clusters of factors the system:

- considers only those combinations for which there is at least one most-on-point, real precedent case that has that combination.
- temporarily ignores the fact that the most-on-point cases, associated with a particular combination of factors, may differ among themselves as to other factors that they do not share with the problem situation.
- temporarily ignores differences in the magnitudes of the shared factors among the most-on-point cases and the problem situation. [Rissland88 p 335]

In CMA these implicit heuristics can instead be explicitly phrased as matching concepts and then training will tell whether they are correct or whether the assumptions they make are actually important. The heuristics in HYPO are instances of general matching knowledge. CMA's shell can determine the effects of all aspects of general matching knowledge of a domain. This makes CMA more flexible and robust because it is less dependent on using hard-coded domain specific knowledge.

The other crucial and novel extension to the processing model in CMA is to provide the system with a training stage. During the training stage a feedback loop is incorporated into the processing model. The system learns definitions of concepts and the importance of its knowledge in different contexts of the domain which help it to give better responses. This way of learning is very different from other CBR systems in two respects. First, classic systems learn by failing during normal use, that is, they do not use a training phase when both positive and negative feedback are given. Second, failure driven learning in classic systems results in modification of the cases themselves. This contrasts to CMA where the system is learns how to better use its cases rather than how to change them.

In CHEF [Hammond86b], after the system is told that the recipe it has formulated is inadequate, the system forms an explanation for the failure and then replans from scratch with the new information. The explanation the system formulates is causal and requires a set of background knowledge about ingredients, their preparation and cooking methods. The system then uses the explanation to modify the case it used so that the

incorrect expectation can be anticipated and avoided in future similar circumstances. This is termed failure-driven learning because learning takes place when the system has failed.

In CMA, learning takes place during a training stage and is both failure-driven and success-driven. A training phase implies that many repairs will be needed when a system is young and just starting to be used (i.e. it has not learned much). This seems to be how people learn; initially they make lots of mistakes and have to be given lots of positive and negative feedback. Using positive feedback is pedagogically sounder. Good teaching emphasizes telling the students what they have done well more than telling them what they have done wrong.

Classic CBR systems need a lot of knowledge before they can learn. They can not learn without being supplied, by hand, with large amounts of knowledge formulated by experts. In contrast, CMA starts to learn from less knowledge and its initial knowledge base does not have to be precisely or accurately defined. CMA takes responsibility by itself to learning a great deal of initial knowledge concerning the domain. The system can improve its own performance simply by subsequently being told that what it supplied was good, bad or indifferent advice.

CMA does not rely on an explicit set of background knowledge to form causal explanations for its failures. This would be useful (background knowledge could be used to generate suggestions due to causal analogy) but such knowledge may not be available in the domains it is designed to handle. It does the next best thing by trying to infer which of its knowledge must be changed to result in a better response the next time a similar situation is encountered.

The approach taken in CMA also differs from example driven programming or example based learning. In this style of knowledge acquisition the expert presents examples directly to the system and it induces a set of rules from a background theory. The representation language strongly influences (decides?) the way the final concepts can be expressed. The interpretation of the predicates is provided by the background knowledge. In CMA, using graphs implies that concept definitions can arise from almost uninterpreted graph theory. This does provide unrestricted expression to the system for its learning. The nature of the result of learning in CMA is somewhat weaker than example driven systems because it is expressed in terms of parameters or strength values rather than as a fully interpreted concept from a background theory. CMA uses experts to their fullest advantage. They can supply an overall approach and pinpoint focus points for the system initially and then keep it in the right direction. This prevents the system from being limited to produce concepts which are too narrow, that is, only cover the given

examples. Since the experts named the concepts this might make them accept the system more readily. This also helps the system give better explanations. The system can use these named concepts to explain why a case was retrieved and why it chose to use a certain part of a case to generate a suggestion. Case-based explanation is typically very useful because it is easy for users to look at the case the system used and to see where a solution came from and judge its validity.

In summary, CMA groups knowledge in terms of when and how it is used by the CBR model. This makes it unique. The fact that it modifies, by itself, the knowledge which interprets cases rather than cases themselves also makes CMA different from classic CBR systems. To our knowledge CMA is the first CBR system to use a training stage for knowledge acquisition.

5.8 CONCEPT DESCRIPTION

In terms of knowledge the system must acquire, two issues are important. First, so that model is general, the representation must work within the shell component and have a translation to application vocabulary. This allows the system to communicate with its users. This is done in our extended model by using graphs as a representation for a case and then using their geometric abstractions to express semantic domain concepts. The system's heuristics express how concepts in the domain interact to determine suitable approaches in finding solutions. Chapter 3 describes this in detail. See in particular Figure 3.2 'Graph Supported Knowledge'. The system only has to examine the type of a heuristic (parameter-based or importance-based) and its classification (appraisal, match quality or suggestion generation) to use it. It does not have to interpret the meaning. This extends the power of the shell and contributes to the generality of the system.

Second, the representation must have a way of describing the domain's characteristics as they pertain to meaningful input analysis, matching and suggesting. One requirement is to be able to describe a concept. Our model makes use of parameters which form ranges in the definition of a concept. A concept will be either used for appraisal, matching or suggestion. For example, in CMA a 'long abstraction diagram' may be defined as one which has a taxonomic path which is longer than 3 relationships and not more than 5. The value of 'long abstraction diagram' is a range between 3 and 5. When matching, a taxonomic path of length 3 may not be very different from a taxonomic path of length 4 for the intents and purposes of building good diagrams. This concept might also be useful for input analysis. If a diagram does not contain a 'long abstraction

diagram' then it may be likely that taxonomic relational elaboration is appropriate. Should this ever turn out to be different, feedback will indicate to the system that it should adjust either the upper or lower bound of the range or its strength. This is essentially what the system does to refine a concept, it adjusts the range for its definition or its importance. The most fine-grained effect of feedback is on an individual basis for each concept. However, while a feedback behavior is specified for a concept, it could actually take the behaviour of other concepts on this occasion or past occasions into account when an adjustment is made to form meta-level generalizations.

Another requirement of the representation is that it facilitates defining whether knowledge is important in a certain context. In matching, this allows some fields to assume more importance than others and thus matching on these field will have a certain effect on the total match. A similar situation occurs during input analysis when the system needs to decide what is the best strategy it should follow for first-level retrieval. In CMA importance-based heuristics return a numeric value which indicates their importance. The strength a concept has in a certain context is defined within the concept itself. For example, an importance-based concept may state that two cases are similar if they both have a cycle of length 4 and that they are even more similar if both cycles have the same kind of relationships. If a cycle exists in neither the input or a case, this concept during matching might 'inform the matching process' that it abstains from being a factor in the matching evaluation. If only one of the input or case has a cycle the concept may return a negative value which decreases the quality of the match. If both have a cycle but not with the same types of relationships a positive strength which is less than the one returned for cycles with the same type of relationships is claimed. The system uses strengths in its calculations which determine what appraisal is the most accurate as well as which match is the best. Note that a concept itself specifies how it contributes to this calculation. The feedback from the expert is used by the concept to make adjustments towards correcting a fault in its strength determination. Again, the most fine-grained effect of feedback is on an individual basis for each concept.

The knowledge acquisition approach our extended CBR model takes differs from that proposed by Riesbeck because it is forced to face problems Riesbeck has assumed will not be present. Riesbeck has proposed ways to help experts express the domain knowledge but they must still formulate the knowledge exactly and precisely as rules. Our model attempts to be robust in domains which are less constrained, quite subjective, and comprising expertise which is hard to formulate as rules. In these domains experts can not necessarily express how to do something well but they can describe what makes

something good. They are able to judge characteristics better than they can consistently justify and explain their methods. Our approach is for the system to use the expert as a source to supply initial, approximate, overall aspects that might be productive for input analysis, matching and suggestion generation. The system derives the uncoded theory that a skilled practitioner has acquired. The theory is expressed, not as rules, but in terms of heuristics which use either concepts that have been refined as the system observes which strategies are useful, or have an importance associated with them. During its training period the system tries a large variety of options and, from the expert's judgement of the resulting output, it can learn relationships between options and results.

5.9 WHAT THE SYSTEM DOES

CMA ultimately learns how to build good diagrams. In general, our extended CBR model can learn the expertise of a skilled practitioner who develops designs in some domain. The model's theory is not couched in terms of a structured methodology or a set of guidelines or fundamental principles. Instead, it is in terms of how to analyze incomplete designs to determine what next should be added to them, how to match incomplete designs with similar completed designs and how to interpret completed designs so that they can be used as a source of analogy. Previous work in goal and plan recognition and machine learning has had the same goal in mind but the approaches taken made the assumption that the expertise the system was trying to acquire could be expressed in a structured set of goals, plans or using a language of operational terms. This is suspect because there are many design domains which are loosely constrained, unstructured and highly creative in nature which prevents their theory from being properly codified. Our approach differs by being more pragmatic. We know that experts rely on their experience to help them solve new problems. We exploit this to avoid contriving a domain theory. Instead, using CBR paradigm, the expertise can be expressed in terms of case-based operations: input analysis, matching, and analogy driven suggestion.

6. FUTURE RESEARCH AND SUMMARY

6.1 FUTURE RESEARCH

A system based on our extended model would be greatly enhanced if it could discover, entirely on its own, either domain specific heuristics or new concepts. One way of learning new heuristics is as follows. Generalizations of existing heuristics could be formed. For example, from 'similar by dependency graphs' the system could generalize to 'similar to a graph theoretical property' and then specialize this to another instance of a graph theoretical property such as 'similar by cycles'. This is case-based inference by derivational analogy. A new heuristic would be instantiated with low probability for any unassessed graph theoretical property and with a higher probability for a property that is useful in a related or previously tried domain. New suggestion heuristics could be obtained by suggesting adding something to the input diagram that would improve the resulting diagram's match quality with the case. Background knowledge could be added to the system in domains where it is obtainable and explanation-based machine learning techniques could be used to obtain generalizations. The system already maintains performance records (for feedback use) and these could be used by inductive learning methods.

Our feedback mechanism associates a feedback procedure with each heuristic. This is very powerful. If a consistent manner to learn heuristics which need both definitional refinement and importance refinement could be found, our model would be more powerful. More experimentation with different feedback schemes for heuristic adjustment remains to be performed. In particular, more experimentation would reveal what is the most appropriate manner for a system to learn its thresholds for changing importance or definitions. The optimal amount of history to remember has not been found nor is it known what scheme best handles noisy domains. Ultimately we would like to introduce consistency monitoring into the extended model so that the system would monitor the changes it makes to its knowledge and be able to go to the expert when it detects an inconsistency. The system would be able to detect problems in the progress of converging to a "good" set of heuristics or oscillation of definitions. We envision a

reason maintenance component for case-based learning analogous to those in rule-based systems [Doyle79].

Han has shown that graphs are a suitable representation for program description for the purposes of Prolog compilation optimization: 'Complex function-free linear recursions are represented with a variable connection graph, the V-graph. Using the variable connection graph, linear recursions are classified into six classes: acyclic paths, unit cycles, uniform cycles, nonuniform cycles, connected components, and their disjoint mixtures. Recursions in each class share some common properties in compilation' [Han89]. In the future we would like to apply CMA to programming. It will be interesting to discover which geometric abstractions arise from the graph interpretations of program pieces to be useful interpretations of the program's function, style and constraints.

CMA is only a prototype knowledge-based assistant. Much implementation and work remains to be done to make it a robust production-quality system. It is possible to envision the users highlighting portions of their diagrams to indicate what portion the system should focus on. This would circumvent the crude approach used at present which focuses on the component associated with the last relational assertion. To be acceptable to professionals the system needs to be integrated completely with its underlying tool MODELLER. Then when CMA actually suggests assertion the user can select the one desired and have it automatically put into the diagram.

6.2 SUMMARY

In summary, we have described the components and processes of the case-based paradigm with the aid of illustrations from classic case-based systems and CMA. The approach of this thesis has been to extend CBR in directions which are, in general, profitable to CBR while meeting the specific purpose of providing knowledge-based assistance. For example, we investigated and used graphs to represent cases. This directly helped to solve a problem which exists in design domains; input is an incomplete case.

To go over the main points again, this thesis makes the following contributions to CBR:

- CBR is used in a novel way as a model to provide knowledge-based assistance (KBA). Traditional CBR systems perform classification or planning while CMA participates interactively with its users to provide suggestions for completing their tasks.

- An approach for KBA in design domains is possible through the use of CBR. Traditional rule-based methods are inadequate to capture the unstructured, unconstrained, creative nature of these domains. Expert designers use natural reminding processes when constructing new designs. CBR suitably expresses this approach.

- Graphs are used to represent cases. They are an excellent representation in assistant systems where a case is a finalized design from which a theory of how to build a 'good' design needs to be formulated. Graphs express data in a form which flexibly permits a wide variety of interpretations. In fact, it is this interpretation of the case which is more important than the finalized design. *A good representation should make important facts explicit and suppress irrelevant detail. It should expose constraints and be perspicuous* [Winston80]. A graph permits auxiliary knowledge to be formulated which expresses what facts contained in it are important. The representation itself stays stable but supports changing interpretations which can be determined by a system on its own

- Domain expert are used as 'oracles' which places only minimal reliance on them to be articulate and to initially supply the system with a large body of correct and precisely formulated knowledge. This form of knowledge acquisition enlarges the number of domains which can use CBR to include those which are currently excluded because their experts can not express their expertise exactly.

Our extended model presents an original approach to automated knowledge acquisition and machine learning. It enhances a CBR system because it can:

- by using graphs, accept an incomplete design as input and find matches for it among finalized designs.

- learn and refine heuristics using an initial amount of approximate knowledge. The system can refine its definition of concepts which are used by heuristics and it can adjust the importance of heuristics according to evidence.

- influence the match quality of retrieved cases by reasoning about context and purpose. The system does not use a static list of match criteria nor does it use a a priori established weighting scheme for importance. Instead, it can determine which heuristics express important correspondences and it can decide on a weight which takes the current situation and what a case will be used for into account.

- adjust its performance at the usually fixed stages of CBR, (analysis , matching and using a case) by employing feedback. The system can improve its analysis, matching, and suggesting by using successful techniques more often and improving less successful techniques.

- extract a 'weak' theory of how to create good designs. This theory is couched in CBR terms rather than as a formal methodology. It makes KBA practical in domains where experts have uncodified knowledge beyond textbook skills which they have obtained from experience.

The extensions are illustrated with examples from a prototype system, CMA, which provides KBA for an Entity-Relationship Diagram design tool. The power of our design is not confined to Entity-Relationship modelling because the model facilitates systems which, in general, are less dependent on receiving a large portion of their knowledge base from an expert prior to their initial operation. We have made a crucial role reversal. Instead of the experts testing the system and changing it to behave the way they desire they, now, they influence the system to make the changes to itself.

Our conclusion is that CBR is an adequate processing paradigm for assistant systems. An extended CBR model furnishes excellent solutions with regard to knowledge acquisition, performance, flexibility, and generality .

A.1 A DESCRIPTION OF CMA'S OPERATION

This is an description of the sequence of events that take place during CMA's operation.

Initial Conditions:

The case library contains completed diagrams which have been parsed and are represented as graphs. The graphs have been preprocessed to determine taxonomic, generic, and dependency subgraphs, connected components, and connection related data. The library stores each case under a name index. Each graph contains indices to its subgraphs, connected components and connection data.

Step 1: Input

The assertions which comprise the diagram under construction with MODELLER are read by CMA and CMA forms a graph of the diagram. It would consist of some number of nodes each with a set of associated incoming and outgoing edges. The nodes represent entities and have associated properties of entities (independence, synonymity). The edges represent relationships and can be of 6 types: *'is Related to'*, *'is Natural Subclass of'*, *'is Role of'*, *'is Characteristic of'*, *'is Association of'*, *'is an Extension of'*. Relationships are directed and have a name, attributes, minimum cardinality, and maximum cardinality.

Step 2: Preprocessing

The input diagram is preprocessed to locate a focus cluster. This is a subgraph which is the largest connected component containing the edge which represents the most recently defined relationship in the input diagram. The focus cluster may be preprocessed to locate its subgraphs by taxonomy or dependence and its connection data.

Step 3: Input Analysis

Does the input diagram require relational elaboration, descriptive elaboration or interconnection? A group of heuristics for each possibility is tested with the input diagram. Each heuristic embodies some use of a domain concept and is either parameter-based or importance-based. A heuristic returns a predicate result which indicates whether

its meaning is present in the input diagram and thus indicates whether the type of elaboration it describes is appropriate. The system calculates a measure for the analysis. Parameter-based heuristics have weight 1 when they return true and 0 when they return false. If they return an 'abstain' result they are not used in the calculation. Importance-based heuristics return their weight or 'abstain'. The sum of the weights is normalized over the total possible weight of the used heuristics. We present some examples of heuristics for each type of analysis. Each heuristic keeps a record of its performance - what it responded, whether it was rewarded and what the value it considered was. The amount of history and its nature depends on what knowledge the feedback procedure needs.

Relational Elaboration: 'The diagram requires more relational information'

Name: 'Not enough taxonomic and association information',

Type: Importance-based

If the focus cluster contains no taxonomic or dependency subgraphs, return strength = 2, otherwise return strength = 0.

Feedback: Adjust strength returned upwards (downwards) whenever this heuristic is involved in the system receiving a reward (penalty) for a relational assertion suggestion. If this heuristic was not present in the input diagram (i.e. returned strength = 0) and a relational assertion was rewarded then it is of less importance so decrease its strength. The adjustment is determined by multiplying the ratio of failed behavior to total responses times a standard increment. For example, if the heuristic had failed 4 times out of 6 and the standard increment was 1.0 then the adjustment would be 0.67.

Name: 'Enough Cardinality Information',

Type: Parameter-Based

If the ratio of edges with cardinalities specified to total edges is greater than 0.5 return true, else return false.

Feedback: 4 situations may occur. A and B indicate the heuristic is performing well. C and D indicate that the ratio value needs to be improved. The current ratio is 0.5.

A) 'Enough Cardinality Information' returns true and results in a rewarded suggestion. The result is to tally a 'good' response.

B) '*Enough Cardinality Information*' returns false and a penalized suggestion is for relational elaboration. The result is to tally a 'good' response.

C) '*Enough Cardinality Information*' returns true and results in a penalized suggestion. The result is to tally a 'bad' response.

D) '*Enough Cardinality Information*' returns false and a rewarded suggestion is for relational elaboration. The result is to tally a 'bad' response.

In both C and D, adjust ratio in appropriate direction by an increment obtained using the ratio of bad responses to good responses times a standard increment.

Descriptive Elaboration: 'The diagram requires more information on the relationships and entities it presently has, in the form of attributes, cardinalities or domains.'

Name: '*Few Cardinalities*',

Type: Parameter-Based

If the ratio of edges to edges with values is less than 0.5 return true, else false.

Feedback: Same as '*Enough Cardinality Information*' feedback.

Name: '*Few Generic Cardinalities*',

Type: Parameter-Based

If the ratio of edges in the generic subgraph to edges with values in the generic subgraph is less than 0.5 return true, else false.

Feedback: Same as '*Enough Cardinality Information*' feedback.

Interconnection: 'The diagram requires two separate components to be linked together.'

Name: '*More than 1 CC*',

Type: Importance-based

If there is another component which is not connected to the focus cluster return 2, else -1.

Feedback: Adjust strength returned upwards (downwards) whenever this heuristic is involved in the system receiving a reward (penalty) for a relational assertion suggestion. If this heuristic was not present in the input diagram (i.e. returned strength = 0) and a relational assertion was rewarded then it is of less importance so decrease its strength. The adjustment is determined by multiplying the ratio of failed behavior to total responses times a standard increment. For example, if the

heuristic had failed 4 times out of 6 and the standard increment was 1.0 then the adjustment would be 0.67.

When the system is finished training only the best analysis measure would direct the first-level retrieval. That is, if the score for descriptive elaboration was higher than that of relational elaboration then the system would retrieve cases with connected components of the same number of nodes as the focus cluster instead of ones with one extra node. For interconnection the system retrieves cases which have connected components of approximately the same size as those in the input diagram with links connecting them. During training the analysis measure is not used since a retrieval based on each of the three analyses is done.

Step 4: Retrieval

As a result of analyzing the input, the system performs its first-level case retrieval. Each case in each of the retrieval sets is compared to the input diagram for similarity. We term the case and input diagram the *candidate pair*. The system has one collection of matching heuristics. Each heuristic is either parameter-based or importance-based. A parameter to the heuristic predicate is the type of analysis (relational elaboration, descriptive elaboration or interconnection). Each heuristic returns a value which depends on the type of analysis which resulted in the case retrieval and each heuristic has a range of values or strength for each type of analysis. This is how the predicate can take the intended use of the case into account in its assessment of the match factor it considers. All heuristics are queried for each candidate pair. The overall strength calculation is a total of the strengths returned by the heuristics normalized by the number of heuristics which were queried and did not abstain. This calculation is 'fixed' in the system but the values used in the calculation are not since they are the values returned by the heuristics and updated by the feedback procedures. Some examples of match quality heuristics are:

Name: ' *Similar by types of relationships* ' ,

Type: Parameter-Based **Values:** Relational: 80%, Descriptive: 90%, Inter: 75%

Find the most accurate isomorphism for the candidate pair. The type of a relationship is indicated by a specific value. The accuracy of the isomorphism is measured as the percentage of equivalent values. If the accuracy exceeds the value corresponding to the analysis type return true, else false. Record the best

isomorphism for mapping purposes later when generating suggestions. Since the input diagram focus cluster has one less node than the case component in relational description retrieval, when testing for isomorphism the system designates one node as extra and excludes it from the matrix comparison.

Feedback: Same as '*Enough Cardinality Information*' feedback but only the value of the appropriate analysis is adjusted.

Name: '*Loosely Similar by types of relationships*' ,

Type: Parameter-Based **Values:** Relational: 90%, Descriptive: 95%, Inter: 85%

Find the most accurate isomorphism for the candidate pair. The type of a relationship is indicated by a specific value. The accuracy of the isomorphism is measured as the percentage of matching values. Two edges match if they are the same type or one of them is a '*is Related To*' relation or the case has a relation but the input diagram does not. If the accuracy exceeds the value corresponding to the analysis type return true, else false. Record the best isomorphism for mapping purposes later when generating suggestions.

Feedback: Same as '*Enough Cardinality Information*' feedback but only the value of the appropriate analysis is adjusted.

Name: '*Similar by number of relationships*' ,

Type: Parameter-Based **Values:** Relational: 2, Descriptive: 0, Inter: 1

Compare the number of explicit relationships for the candidate pair. The number may be the same within the range stipulated by value. If they are return true, else return false.

Feedback: Same as '*Enough Cardinality Information*' feedback but only the value of the appropriate analysis is adjusted.

Name: '*Similar by existence of dependency subgraphs*' ,

Type: Importance-based **Values:** Relational: (1,0), Descriptive: (1,0), Inter: (1,0)

For relational and descriptive elaboration purposes, return first value in the pair if both graphs in the candidate pair have a dependency subgraph, second value in pair if only one graph has a dependency subgraph and abstain in neither have a

dependency subgraph. When interconnection is the intended use of the case, return the first value if they both have a dependency subgraph or one has a dependency subgraph. Otherwise abstain.

Feedback: Same as '*More than 1 CC*' feedback but only the value of the appropriate analysis is adjusted.

'Similar by existence of dependency subgraphs' can be appropriately reformulated to consider taxonomic subgraphs, cycles of a given size, cycles of any size, equivalent radii, equivalent diameters or both taxonomic and dependency subgraphs. Each heuristic keeps a record of its performance - what it responded, whether it was rewarded and what the value it considered was for each type of analysis. The amount of history and its nature depends on what knowledge the feedback procedure needs. The feedback procedure could be designed to adjust values differently according to the type of analysis the result pertained to.

Each candidate pair ends up with a match quality score which is determined by the values returned by the match quality heuristics. Outside training only the candidate pair which the system considers to be the best would be input to the suggestion phase. During training the highest 'n' best matches are used as input data in the suggestion phase.

Step 5: Suggestion Generation

The intention in this phase is to find out what information in the case is valuable as a suggestion and then to map this suggestion onto the input diagram. In the cases where accurate isomorphisms have been found these supply the mappings. Otherwise other (quicker) approximate mappings can be formed using heuristics which match centers and nodes at the same distance from the center of a component. The outcome of suggestion generation can be one assertion or a set of compound assertions. In CMA all suggestion heuristics are importance-based. The strengths indicate the quality of the suggestion strategy. There is no overall strength calculation. Instead the strengths will be used post-training to rank the strategies and generate only the assertions from the best ones. Like the heuristics used at analysis and matching, suggestion heuristics are also classified according to analysis purpose. The feedback procedure is very simple and is the same as '*More than 1 CC*'. Examples of suggestion heuristics follow:

Relational Elaboration: 'The diagram requires more relational information'

Name: 'Recommend new relationship',

Type: Importance-based

For the most accurate mapping obtainable, recommend every extra relationship in the case which does not involve adding a new entity.

E.G. 'Producer is Related to Product' where producer and product are already existing entities but with no relationship between them.

Name: 'Recommend new relationship and entity',

Type: Importance-based

For the most accurate mapping obtainable, recommend every extra relationship in the case which involves adding a new entity linked to an existing entity.

E.G. 'Producer is a Characteristic of NEW-ENTITY' where producer already exists as an entity.

Descriptive Elaboration: 'The diagram requires more information on the relationships and entities it presently has, in the form of attributes, cardinalities or domains.

Name: 'Recommend explicit cardinalities for is Related to relationships',

Type: Importance-based

For the most accurate mapping obtainable, recommend the same minimum and maximum cardinality values which exists for 'is Related to' edges in the case for corresponding edges without cardinalities in the input diagram.

E.G. 'Min cardinality of producer to product is 0' and 'Max cardinality of producer to product is n' where producer and product are already existing entities with an 'is Related to' relationship between them.

Name: 'Recommend unspecified cardinalities for is Related to relationships',

Type: Importance-based

For the most accurate mapping obtainable, recommend the user add minimum and maximum cardinality values for 'is Related to' edges which exist in the case and have corresponding edges without cardinalities in the input diagram.

E.G. 'Min cardinality of producer to product is ?' and 'Max cardinality of producer to product is ?' where producer and product are already existing entities with an '*is Related to*' relationship between them.

A number of variations on these heuristics can be generated by exchanging the mapping used, the specificity supplied in the suggestion, and the type of relationship considered.

Interconnection: 'The diagram requires two separate components to be linked together.'

Name: 'Connect the two components at their centers',

Type: Importance-based

Using the links which connect the 2 connected components in the case, suggest adding a corresponding link which involve nodes closest to the center of each component.

E.G. 'producer is a role of vendor' where producer and vendor are already existing entities in unconnected components.

A number of variations on interconnection heuristics can be generated by exchanging the links used and the specificity supplied in the suggestion.

If heuristics produce the same assertion the system only shows one but relays the oracle's feedback to each heuristic that generated the assertion. A generalizing feedback procedure can be incorporated which changes the heuristics to be more general if it observes a related set all behaving effectively.

Step 6: Display and Feedback Reception

The assertions which constitute suggestions are assembled and displayed in a 'Training Browser'. The experts can see all assertions the system considers plausible according to the knowledge it has at this point in time. Each assertion may be selected and a pop-up menu allows it to be rewarded or penalized. The experts can also select the 'How' option in the pop-up menu which presents them with an 'Explanation Browser' that shows how the system came to recommending the selected assertion. In the Explanation Browser the results of analysis are shown, the candidate pairs and match quality results which produced the suggestion are available for perusal and the suggestion heuristics which directly formed the assertion are also present. The system also has

facilities which enable the user to view the input diagram in its graph form, to see the cases in graph form in the library, and to set up the training data for initialization and use in on going training. A transcript runs interactively with the system as it processes to show the user what the system is doing. It can be restricted to a less verbose mode. Heuristics are actual objects in the system and they can be inspected to review their predicates, feedback procedures, historical data, and current values.

Step 7: Feedback

The feedback loop works because the system keeps a trace from every assertion back to:

- 1) each suggestion heuristic which generated it
- 2) each candidate pair and the match quality heuristic results which were input to each of the suggestion heuristics in (1).
- 3) the input analysis heuristic results for the type of analysis which was assumed in matching and suggesting.

Each feedback procedure receives two parameters: the feedback result (penalty or reward) and a data structure termed a 'suggestion record' of related data which can be used to retrieve information related to history, overall performance or input context.

REFERENCES

- [Aho83] Aho, A. V., Hopcroft, J. E. and Ullman, J. D., Data Structures and Algorithms, Addison-Wesley Publishing Co., 1983
- [Alterman88] Alterman, R., 'An Adaptive Planner' in Proceedings: Case-Based Reasoning Workshop, (37-49), Morgan Kaufmann, 1988
- [Ashley88] Ashley, K., Rissland, E., 'Compare and Contrast, a Test of Expertise' in Proceedings: Case-Based Reasoning Workshop, (31 - 36), Morgan Kaufmann, 1988
- [Bain85] Bain, W. M., 'A Case-Based Reasoning System For Subject Assessment' in Seventh International Joint Conference on Artificial Intelligence, (523-527), 1985
- [Bain86] Bain, W. M., Case-Based Reasoning: A Computer Model of Subjective Assessment, Ph.D. Thesis, Yale University, 1986
- [Bareiss89a] Bareiss, R., 'The Experimental Evaluation of a Case-Based Learning Apprentice' in Proceedings: Case-Based Reasoning Workshop, (162-167), Morgan Kaufmann, 1989
- [Bareiss89b] Bareiss, R., King, J. A., 'Similarity Assessment in Case-Based Reasoning' in Proceedings: Case-Based Reasoning Workshop, (67-71), Morgan Kaufmann, 1989
- [Barletta88] Barletta, R., Mark, W., 'Explanation-Based Indexing of Cases' in Proceedings: Case-Based Reasoning Workshop, (50 - 60), Morgan Kaufmann, 1988
- [Beineke78] Beineke, L. W., Wilson, R. J., (Eds), Selected Topics in Graph Theory, Academic Press, 1978
- [Borning87] Borning, A., 'Constraint Hierarchies', Proceedings of OOPSLA '87, pp 48-60, (1987)
- [Carberry89] Carberry, S., 'Plan Recognition and its Use in Understanding Dialog' in User Models in Dialog Systems, (Chapter 6), Springer-Verlag, 1989
- [Carbonell88] Carbonell, J., Veloso, M., 'Integrating Derivational Analogy into a General Problem Solving Architecture' in Proceedings: Case-Based Reasoning Workshop, (104-124), Morgan Kaufmann, 1988
- [Carver84] Carver, N. F., Lesser, V. R. and McCue, D. C., 'Focusing in Plan Recognition' in Proceedings of AAAI, (42-48), 1984

-
- [Chachra79] Chachra, V., Ghare, P. M. and Moore, J. M., Applications of Graph Theory Algorithms, Elsevier North Holland Inc, 1979
- [Chin89] Chin, D., N., 'KNOME: Modelling What The User Knows in UC' in User Models in Dialog Systems, (Chapter 4), Springer-Verlag, 1989
- [Constant90] Constant, P., Matwin, S. and Oppacher, F., 'LEW: Learning by Watching', I.E.E.E. Transactions on Pattern Analysis and Machine Intelligence, Vol 12, 3, pp 294-308, (1990)
- [Deugo89] Deugo, D. L., Oppacher, F., 'Case-Based Techniques for Learning and Repairing Plans' in Proceedings of the Second International Symposium on Artificial Intelligence, 1989
- [Doyle79] Doyle, J., 'A Truth Maintenance System', Artificial Intelligence, Vol 12, pp 231-272, (1979)
- [Dupuy88] Dupuy, C. T. N., 'Military History and Case-Based Reasoning' in Proceedings: Case-Based Reasoning Workshop, (125-135), Morgan Kaufmann, 1988
- [Fikes71] Fikes, R. E., Nilsson, N. J., 'STRIPS: A new approach to the application of theorem proving to problem solving', Artificial Intelligence, Vol 2, pp 189-208, (1971)
- [Ford90] Ford, K. M., Adams-Webber, J. R., 'Knowledge Acquisition and Constructivist Epistemology' in The Cognitive Psychology of Expertise: Empirical Approaches to Knowledge Acquisition, Springer-Verlag, 1990
- [Freeman90] Freeman-Benson, B., 'An Incremental Constraint Solver', Communications of the ACM, Vol 33, 1, pp 54-63, (1990)
- [Gondran84] Gondran, M., Minoux, M., Graphs and Algorithms, Wiley-Interscience Publications, 1984
- [Hammond86a] Hammond, K. J., 'Case-Based Planning : An Integrated Theory of Planning, Learning, and Memory' in Ph.D. thesis, 1986
- [Hammond86b] Hammond, K. J., 'CHEF: A Model of Case-Based Planning' in Proceedings of the Fifth National Conference of Artificial Intelligence, (267-271), Morgan Kaufmann, 1986
- [Hammond88] Hammond, K. J., 'Case-Based Planning' in Proceedings: Case-Based Reasoning Workshop, (17-20), Morgan Kaufmann, 1988
- [Hammond89a] Hammond, K. J., 'Adaptation of Cases' in Proceedings: Case-Based Reasoning Workshop, (88-89), Morgan Kaufmann, 1989
- [Hammond89b] Hammond, K. J., Case-Based Planning: Viewing Planning as a Memory Task, Academic Press, 1989

- [Han89] Han, J., 'Compiling general linear recursions by variable connection graph analysis', *Journal of Computational Intelligence*, Vol 5, pp 12-31, (1989)
- [King88] King, J. A., Klein, G. A., Whitaker, L. and Wiggans, S., 'SURVER III: An Application of Case-Based Reasoning' in *Proceedings of the Fourth Annual Aerospace Applications of AI Conference*, 1988
- [Klein88a] Klein, G. A., Calderwood, R., 'How Do People Use Analogues to Make Decisions?' in *Proceedings: Case-Based Reasoning Workshop*, (209-223), Morgan Kaufmann, 1988
- [Klein88b] Klein, G. A., Whitaker, L. A. and King, J. A., 'Using Analogues To Predict and Plan' in *Proceedings: Case-Based Reasoning Workshop*, (224-232), Morgan Kaufmann, 1988
- [Kobsa89] Kobsa, A., 'A Taxonomy of Beliefs and Goals for Use Models in Dialog Systems' in *User Models in Dialog Systems*, (Chapter 3), Springer-Verlag, 1989
- [Kolodner84] Kolodner, J. L., *Retrieval and Organizational Strategies in Conceptual Memory: A Computer Model*, Lawrence Erlbaum Associates, 1984
- [Kolodner88a] Kolodner, J. L., 'Extending Problem Solving Capabilities Through Case-Based Inference' in *Proceedings: Case-Based Reasoning Workshop*, (21-30), Morgan Kaufmann, 1988
- [Kolodner88b] Kolodner, J. L., 'Retrieving Event from a Case Memory: A Parallel Implementation' in *Proceedings: Case-Based Reasoning Workshop*, (233-249), Morgan Kaufmann, 1988
- [Kolodner86] Kolodner, J. L., C.K., R., (Eds), *Experience, Memory, and Reasoning*, Lawrence Erlbaum Associates, 1986
- [Kolodner87] Kolodner, J. L., Simpson, K. and Sycara-Cyranski, K., 'A Process Model of Case-Based Reasoning in Problem Solving' in *Proceedings of the Sixth National Conference of Artificial Intelligence*, (284--290), Morgan Kaufmann, 1986
- [Koton88] Koton, P., 'Reasoning about Evidence in Causal Explanations' in *Proceedings: Case-Based Reasoning Workshop*, (260-270), Morgan Kaufmann, 1988
- [Lebowitz87] Lebowitz, M., 'Experiments with Incremental Concepts Formations: UNIMEM' in *Machine Learning Vol II*, (130-138), 1987
- [Matwin89] Matwin, S., Oppacher, F. and Constant, P., 'Knowledge Acquisition by Incremental Learning from Problem-Solution Pairs', *Journal of Computational Intelligence*, Vol 5, 2, (1989)

- [Navichandra88] Navichandra, D., 'Case-Based Reasoning in CYCLOPS, a Design Problem Solver' in Proceedings: Case-Based Reasoning Workshop, (286-301), Morgan Kaufmann, 1988
- [Oppacher88] Oppacher, F., 'A Dynamic Case-Based Planning System for Space System Applications' in Fourth Conference on Artificial Intelligence for Space Applications, (431-440), NASA Conference Publication Publication 30133, 1988
- [Quilici89] Quilici, A., 'Detecting and Responding to Plan-Oriented Misconceptions' in User Models in Dialog Systems, (Chapter 5), Springer-Verlag, 1989
- [Riesbeck88] Riesbeck, C. K., 'An Interface for Case-Based Knowledge Acquisition' in Proceedings: Case-Based Reasoning Workshop, (312-326), Morgan Kaufmann, 1988
- [Riesbeck89] Riesbeck, C. K., Schank, R. C., Inside Case-based Reasoning, Lawrence Erlbaum Associates, 1989
- [Rissland88] Rissland, E. L., Ashley, K. D., 'Credit Assignment and the Problem of Competing Factors in Case-Based Reasoning' in Proceedings: Case-Based Reasoning Workshop, (327-344), Morgan Kaufmann, 1988
- [Rissland89] Rissland, E. L., Kolodner, J. L. and Waltz, D., 'Case-Based Reasoning' in Proceedings: Case-Based Reasoning Workshop, (1-13), Morgan Kaufmann, 1989
- [Sacerdoti77] Sacerdoti, E. D., A Structure for Plans and Behaviour, Elsevier, New York, 1977
- [Schank82] Schank, R. C., Dynamic Memory - A Theory of Reminding and Learning in Computers and People, Cambridge University Press, Inc, 1982
- [Simpson85] Simpson, R. L., A Computer Model of Case-Based Reasoning in Problem Solving: An Investigation in the Domain of Dispute Mediation, Ph.D. Thesis, Georgia Institute of Technology, 1985
- [Sycara88] Sycara, K., 'Using Case-Based Reasoning for Plan Adaptation and Repair' in Proceedings: Case-Based Reasoning Workshop, (425-434), Morgan Kaufmann, 1988
- [Tauzovich87] Tauzovich, B., Causal Reasoning in a Software Advisor, 1987
- [Tsichritzis82] Tsichritzis, D. C., Lochovsky, F. H., Data Models, Prentice-Hall Inc., 1982
- [Ulrich88] Ulrich, K., Seering, W., 'Computation and Conceptual Design', Robotics and Computer-Integrated Manufacturing, Vol 4, 3/4, pp 309-315, (1988)

-
- [Wilensky83] Wilensky, R., Planning and Understanding: A Computational Approach To Human Reasoning, Addison Wesley, 1983
- [Wilson79] Wilson, R. J., Beineke, L. W., (Eds), Applications of Graph Theory, Academic Press Inc., 1979
- [Winston80] Winston, P. H., 'Learning and Reasoning by Analogy', Communications of the ACM, Vol 23, Number 12, pp 689 - 703, (1980)
- [Woodroffe88] Woodroffe, M., 'Plan Recognition and Intelligent Tutoring Systems' in Artificial Intelligence and Human Learning, 1988
- [Winston80] Winston, P. H., 'Learning and Reasoning by Analogy', Communications of the ACM, Vol 23, Number 12, pp 689 - 703, (1980)

END

1 2 0 4 9 1

FIN