# CASE-BASE MAINTENANCE: THE HUSBANDRY OF EXPERIENCE

David C. Wilson

Submitted to the faculty of the University Graduate School
in partial fulfillment of the requirements
for the degree
Doctor of Philosophy
in the Department of Computer Science
Indiana University

July 2001

Accepted by the Graduate Faculty, Indiana University, in partial fulfillment of the requirements of the degree of Doctor of Philosophy.

Doctoral Committee

David B. Leake, Ph.D.
(Principal Advisor)

Michael Gasser, Ph.D.

Dirk Van Gucht, Ph.D.

December 6, 2000

Randall Bramley, Ph.D.

# Acknowledgements

So very many people have touched my life over the course of this odyssey. For those that may escape mention here—a dissertation in itself—I have made the last to be first, and I perhaps owe you a drink.

Resounding thanks and a chorus of gratitude to my advisor, David Leake, for all of his support, guidance, and encouragement. David has always been available with a cheerful word, whether dispensing encyclopedic knowledge about relevant people and research, providing constructive criticism, wrestling with the deep and fundamental mysteries of CBR, or simply chatting away. He has done his best to keep me out of trouble for the better part of a decade.

I also want to thank the members of my committee, Mike Gasser, Randy Bramley, and Dirk Van Gucht. I very much appreciate their perspective, assistance, advice, and support.

I would like to thank the host of Artificial Intelligence, Cognitive Science, and especially CRANIUM lab members for their helpful ideas, for their support and wisdom, and for the use of their books. They made the lab a comfortable home away from home. Lab members have included: Andy Kinley, Kyle Wagner, Jim Newkirk, Raja Sooriamurthi, Susan Fox, Ryan Scherle, Doug Blank, Lisa Meeden, Fred Cummins, Keiichi Tajima, Doug Eck, Jerry Franke, Paul Kienzle, Peter Drake, Jenett Tillotson, Eliana Colunga, and Cathy Rogers.

A toast to my collaborators on the various research projects and endeavors that have woven their way into this work: David Leake, Andy Kinley, Alberto Cañas, Jim Newkirk, Shannon Bradshaw, Mirjam Minor, Qiang Yang, Barry Smyth, Peter Funk, Thomas Roth-Berghofer, Vikram Subramaniam, Gary Parker, Randy Bramley, Travis Bauer, Ana Maguitman, Arijit Sengupta, and Jing Ma.

A number of places have provided academic succor throughout the business, and I would very much like to thank Kris Hammond and the Infolab group at Northwestern University for giving me a home during David's sabbatical and beyond. My time there provided unique perspective and ideas, as well as excellent lunchtime banter. The

good folk in the Infolab have included: Larry Birnbaum, Jay Budzik, Shannon Bradshaw, Robb Thomas, Josh Flachsbart, David Franklin, Julie Dubiner, Andy Crossen, Marko Krema, Robin Hunicke, Xiaobin Fu, Kurt Fenstermacher, and Cameron Marlow. I would also like to thank Mark Keane and the folk at University College Dublin for their support during the home stretch, and for the craic.

Thanks very much to the wonderful people in the Case-Based Reasoning community that have been very helpful and supportive in the development of this research.

I want to thank my most excellent group of friends and fellow IU graduate students that have been with me from the beginning: Erik Hilsdale and Sue Lato, Kyle Wagner and Amy Boles, and PJ Weingartner. Thanks to Jeremy Frens for helping to keep the rent down, the grill up, and for providing the very best in entertainment. Thanks to Jim Newkirk for just the right sort of diogenean symposia. For those that did not believe it could be done, the phrase "Bite Me!" now appears in an academic publication, and the onus is upon you to find it.

My deepest thanks go to my parents for all their love, support, and encouragement. I also want to thank God for seeing me through, and that it is over.

This work was supported in part by NASA under award No. NCC 2-1035.

# Abstract

Case-based reasoning (CBR) is an artificial intelligence methodology that uses specific encapsulated prior experiences as a basis for reasoning about similar new situations. CBR systems rely on various "knowledge containers," such as the case-base of prior experiences and similarity criteria for comparing situations and retrieving the most relevant cases. Explicit or implicit changes in the reasoning environment, task focus, and user base may influence the fit of the current knowledge state to the task context, which can affect the quality and efficiency of reasoning results. Over time, the knowledge containers may need to be updated in order to maintain or improve performance in response to changes in task or environment. In particular, maintaining the case-base—the traditional mainstay of knowledge underlying CBR systems—is essential for preserving and expanding the capability of a CBR system throughout its life-cycle.

This dissertation provides a first coherent picture of the overall case-base maintenance problem in CBR and develops new case-base maintenance techniques within that paradigm. The thesis presents a theoretical framework for describing case-base maintenance techniques according to the types of maintenance policies implemented by a given system. The framework serves to unify current maintenance practice, to point out areas for new fundamental research, and as a step toward recommending the best maintenance practices for varying system performance goals. In that context, the thesis goes on to make an examination and account of underlying regularity assumptions in the CBR process that directly affect maintenance activity.

The theoretical picture of case-base maintenance is then complemented with a presentation of new methods and experiments in applied case-base maintenance. The thesis first presents DRAMA, a case-based tool developed for aerospace design support at NASA, Ames, which provides facilities for initial case capture and subsequent refinement that directly exploit user knowledge. By monitoring users as they go about normal high-level design tasks, DRAMA automatically captures user design choices and rationale that can be used to provide proactive recommendations at both the design and design component levels. This helps to maintain the case-base through

continuous support for case-authoring and design consistency, while significantly ameliorating the knowledge-engineering burden on system users.

Next, the thesis presents a practical model for transforming case-base implementations in the Metamorphoses project for representational maintenance. The thesis goes on to examine new methods for automatically maintaining case-bases by incorporating explicit performance concerns into measures of case-base competence in order to optimize case-base composition.

Finally, the thesis describes how the work developed for case-base maintenance can generalize across knowledge containers. The framework for case-base maintenance is extended and applied in the general knowledge container context for case-based reasoner maintenance. The thesis describes applied maintenance beyond the case base, and it presents an application of similarity maintenance in the context of the CBMatrix case-based recommender system for problem-solving support in Scientific Computing.

The whole provides a unifying framework and algorithms for constructing and maintaining CBR systems that may be used over extended periods of time and in changing environments—a valuable resource for implementers, maintainers, and users of CBR systems.

# Contents

**Acknowledgements** **iv**

**Abstract** **vi**

**1 Introduction** **1**

1.1 Case-Based Reasoning . . . . . 2

Committing Cases to Memory . . . . . 3

Remembering Relevant Cases . . . . . 4

Applying Remembered Cases . . . . . 4

The CBR Cycle . . . . . 4

CBR Knowledge Containers . . . . . 4

CBR in Practice . . . . . 5

1.2 Case-Base Maintenance . . . . . 5

1.3 A Framework for Case-Base Maintenance . . . . . 6

1.4 Regularity Assumptions in CBR . . . . . 7

1.5 Interactive Maintenance Support . . . . . 7

1.6 Representational Maintenance . . . . . 8

1.7 Performance & Competence Relations . . . . . 9

1.8 Beyond the Case Base . . . . . 9

1.9 Similarity Maintenance . . . . . 10

1.10 Dissertation Overview . . . . . 10

**2 Case-Base Maintenance Framework** **12**

2.1 Defining Case-Base Maintenance . . . . . 13

2.2 CBM Performance Objectives and Constraints . . . . . 14

2.3 A Framework for Describing CBM Policies . . . . . 15

Data collection . . . . . 16

Triggering . . . . . 17

Proactive vs. Reactive Maintenance . . . . . 18

Operation types . . . . . 18

Execution . . . . . 19

Categorizing Policies for Case-Base Maintenance . . . . . 19

Policies targeting domain content . . . . . 19

Policies targeting indices . . . . . 23

Policies targeting maintenance policies . . . . . 24

2.4 Meta-Maintenance by Lazy CBM . . . . . 24

2.5 Trends and Lazy Maintenance . . . . . 25

2.6 Maintenance and Overlapping Knowledge Containers . . . . . 26

**3 Regularity** **28**

3.1 Defining Regularities for CBR . . . . . 29

Basic Assumptions and Definitions . . . . . 29

Defining Problem-Solution Regularity . . . . . 31

Defining Problem-Distribution Regularity . . . . . 32

3.2 Perspective on Regularity-Related Research . . . . . 33

3.3 Calculating the Regularity Values . . . . . 35

3.4 Using the Formulas as Maintenance Triggers . . . . . 36

3.5 Determining How to Respond: The Role of Diachronic Analysis . . . 36

3.6 Tools for Trend Detection . . . . . 38

3.7 Two Examples: Error Trends and Hot Spots . . . . . 38

Addressing Solution Error Trends: . . . . . 38
Addressing Hot Spots . . . . . 40
3.8 Considerations for Costs and Benefits . . . . . 41

**4 DRAMA: Interactive Maintenance Support 43**

4.1 DRAMA Overview . . . . . 44
4.2 The Task Domain . . . . . 45
4.3 Tenets of the Approach . . . . . 46
4.4 Background . . . . . 46
Case-Based Design Support . . . . . 46
Concept Mapping . . . . . 47
Building Concept Maps . . . . . 48
4.5 Benefits of Integrating CMaps and CBR . . . . . 49
4.6 The DRAMA System . . . . . 50
Using CMaps to Organize and Represent Design Information . . . . . 50
How the System Supports Design . . . . . 50
4.7 DRAMA's Method for Retrievals to Support Adaptations . . . . . 56
4.8 Testing DRAMA's Context-Based Retrieval . . . . . 58
Generating Problems and Retrieval Targets . . . . . 58
Simulating the Manual Retrieval Processes . . . . . 59
Experimental Setup . . . . . 61
Experimental Results . . . . . 61
4.9 Perspective on Issues and Methods . . . . . 64
Case Authoring . . . . . 65
Interactive Case Acquisition . . . . . 65
Guiding Design Rationale Capture . . . . . 66
Conversational CBR . . . . . 67
Knowledge Navigation . . . . . 68

**5 Metamorphoses: Representational Maintenance 69**

5.1 Metamorphoses Overview . . . . . 70

5.2 Implementation Models . . . . . 71

5.3 Realizing Implementations . . . . . 72

Enterprise/RDBS . . . . . 73

Web-based/XML . . . . . 74

5.4 Transforming Implementations . . . . . 75

Web-Based → Enterprise . . . . . 76

Web-Based → Task-Based . . . . . 77

Enterprise → Web-Based/Task-Based . . . . . 77

Task-Based → Web-Based/Enterprise . . . . . 77

**6 Competence & Performance 79**

6.1 The Competence-Performance Dichotomy . . . . . 80

6.2 Performance Goals for Case-Base Maintenance . . . . . 81

6.3 The Value of Performance-Based Criteria . . . . . 82

6.4 A Performance-Based Metric for Case Selection . . . . . 84

Competence & the RC Metric . . . . . 84

The RP Metric . . . . . 85

6.5 Experimental Results . . . . . 86

Performance Effects of Competence Coverage Thresholds . . . . . 86

Compressed Size vs. Adaptation Cost Tradeoffs . . . . . 87

CNN, RC-CNN and RP-CNN for Uniform Case Distributions . . . . 88

RC vs. RP Deletion for Non-Uniform Case Distributions . . . . . 89

Performance Benefit Metric . . . . . 90

6.6 Comparison to Previous Research . . . . . 91

**7 Beyond the Case Base** **93**

7.1 A General Framework for Case-Based Reasoner Maintenance . . . . . 94

Extending the CBM Framework to CBRM . . . . . 94

Categorizing Policies for Other Knowledge Containers . . . . . 95

7.2 Coordinating Maintenance Across Knowledge Containers . . . . . 97

Selecting the Container to Maintain . . . . . 97

Managing Interactions Between Knowledge Containers . . . . . 97

Case Knowledge → Adaptation Knowledge . . . . . 98

**8 CBMatrix: Similarity Maintenance** **100**

8.1 CBMatrix Overview . . . . . 100

8.2 Recommenders for Scientific PSEs . . . . . 102

Recommendation Types . . . . . 102

Artificial Intelligence Recommendation Methods for Scientific Computing 103

8.3 CBMatrix . . . . . 105

Data Structure Recommendation . . . . . 105

Refining Similarity Criteria . . . . . 106

8.4 Developing CBMatrix . . . . . 107

**9 Conclusion** **108**

9.1 Maintenance Framework . . . . . 108

9.2 Regularity . . . . . 109

9.3 Interactive Maintenance . . . . . 110

9.4 Representational Maintenance . . . . . 111

9.5 Competence & Performance . . . . . 112

9.6 Case-Based Reasoner Maintenance . . . . . 113

9.7 Similarity Maintenance . . . . . 114

9.8 Dénouement . . . . . 114

# List of Figures


1.1 Standard case-based reasoning cycle. . . . . 3

3.1 Web page accesses by month. . . . . 41

4.1 Sample screen images produced by the CMap tools. . . . . 49

4.2 Steps in DRAMA's case-based design generation. Enumerated points are sequential; bullets are mutually exclusive choices within steps. . . 52

4.3 Beginning derivation of a new design from a prior case. . . . . 53

4.4 Interface for presenting engine suggestions and entering additional constraints to refine ordering by matching against textual rationale information. . . . . 54

4.5 Steps in the manual retrieval process. . . . . 60

4.6 Retrieval quality for 3 sets of perturbations of random magnitudes. . 61

4.7 Efficiency for 3 sets of perturbations of random magnitudes. . . . . 63

4.8 Retrieval quality at successive steps in the design process for 3 sets of perturbations. . . . . 63

5.1 Relating CBR implementation types. . . . . 71

5.2 Entity Relationship diagram for a typical case-based reasoning process. 73

6.1 Three example cases and their coverage. . . . . 83

6.2 Adaptation effort as a function of threshold, for RC-CNN compression. 88

6.3 Case-base size as a function of threshold level, for RC-CNN compression. 89

6.4 Average adaptation effort for non-uniform case distributions. . . . . 90

# List of Tables

2.1 Sample CBM approaches placed along major dimensions . . . . . 20

# 1

# Introduction

People often solve problems based on specific earlier experiences. This can be seen in tasks ranging from cooking a favorite Szechuan stir-fry, to selecting appropriate evening-wear knowing what has previously gone over well, to carefully working out this year's income taxes following last year's favorable results. Sometimes, however, experiences that have worked well under similar circumstances in the past become less suited for current use. When cooking for housemates, a beloved spicy dish of one's own may not be palatable or possible for everyone (consider the very sad case indeed, of hot-pepper allergies). An explicit change in the regular consumers of a meal may necessitate a change in the usual recipe. A wardrobe that has grown fairly large may offer too many choices for an evening's attire, in extreme, delaying the wine and appetizers. Having many options may seem well and good, but the utility of such an abundance can be reduced when the focus is on quick and timely selection. An unnoticed change in the tax code could cause serious dismay at audit time. Implicit variance in the environment, with respect to the person in question, can trigger a serious re-thinking of appropriate tax-deductions. Thus, there comes a time to select a new cookbook, to give away a few outfits, and to adjust tax return expectations. That is, there comes a time to update our experiences and how we employ them in order to maintain their usefulness.

Similar problems of maintaining the usefulness of prior experiences are faced in artificial intelligence systems. Explicit or implicit changes in the users, task focus, or environment of a reasoning system may require an update in the way the system employs its knowledge. Ideally, the system itself should be able to detect or anticipate when such changes are necessary and be able to make the appropriate adjustments. Likewise, intelligent interactive systems should be able to interactively and proactively support their human system maintainers. This dissertation investigates issues in how artificial intelligence systems, in particular case-based reasoning (CBR) systems, maintain the underlying knowledge upon which the systems rely. In case-based

reasoning, the traditional mainstay of reasoning knowledge rests in the case-base, the library of encapsulated specific prior experiences, or *cases*, that are used in similar current situations as a basis for reasoning. This dissertation focuses on the theory and practice of maintaining such experiential case knowledge for case-based reasoning systems; in other words, the husbandry of experience.

Maintaining the underlying case knowledge in case-based reasoning systems is essential for systems to preserve and improve reasoning quality, efficiency, and usefulness. This dissertation addresses the maintenance problem in CBR by:

- defining case-base maintenance and establishing a theoretical framework for maintenance systems,
- developing and analyzing new case-base maintenance techniques, and
- showing that the theory and practice developed for case-base maintenance can be generally applied to other reasoning knowledge containers.

This chapter introduces case-based reasoning and case-base maintenance, gives an overview of the theoretical maintenance issues and practical maintenance work addressed, and outlines the remainder of the dissertation.

## 1.1 Case-Based Reasoning

Case-based reasoning (CBR) is an artificial intelligence methodology that uses specific encapsulated prior experiences or *cases* as a basis for reasoning about similar new situations. CBR has its roots in the observation that people often reason by recalling similar past experiences to address current problems [Schank 1982], and a number of studies support CBR as a cognitive model (e.g., [Ross 1984; Pirolli and Anderson 1985; Ross 1989; Schmidt *et al.* 1990; Faries and Schlossberg 1994]). Consider, for example, writing a new computer program to extract and analyze information from a customer database. The programmer will often, either directly or by consultation, base the new program on code that they (or, perhaps, a colleague) previously put together for a similar analysis. If the fundamentals of access and interface remain the same, only very specific parts pertaining to the requisite data analysis may need to be adapted.

In this dissertation, we are interested in the application of case-based reasoning techniques to various kinds of problem solving. More specifically, we are interested in how to effectively maintain and support the case-based reasoning process itself—a process that, in turn, supports problem solving. There are many different ways of

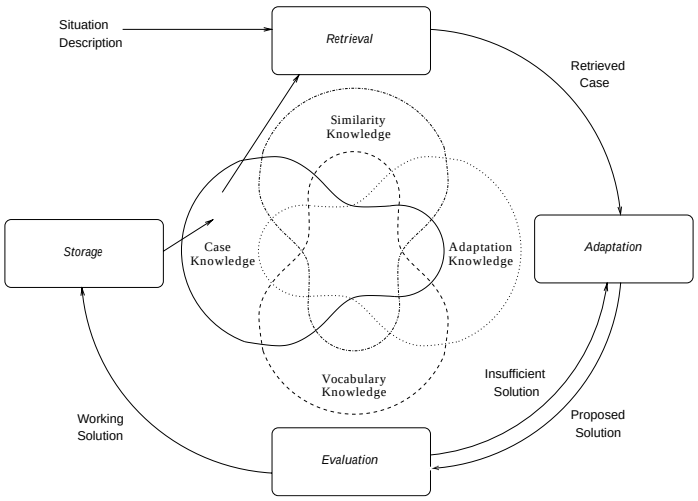



Figure 1.1: Standard case-based reasoning cycle.

describing the case-based reasoning process (e.g., [Kolodner 1993; Aamodt and Plaza 1994; Kolodner and Leake 1996]), but case-based reasoning systems typically embody three important aspects: committing cases to memory, remembering relevant cases, and applying remembered cases. Figure 1.1 shows a representation of the case-based reasoning cycle.

## Committing Cases to Memory

Cases are usually thought of as using a *vocabulary* of attributes, predicates, etc. to represent their encapsulated experiences. In committing experiences to memory, a CBR system identifies the salient features or aspects that best characterize a situation in terms of the vocabulary. Often this set of features, or *indexing vocabulary*, is fixed for a given system. For example, a simple real-estate property characterization might consist of: number of bedrooms, number of bathrooms, area, and price. Case indices are then used in conjunction with a persistent *storage* mechanism for retaining the experiences in memory. Appropriate case *indexing* is a major research area in case-based reasoning (e.g., [Fox 1995]).

## Remembering Relevant Cases

In remembering relevant cases, the salient features of the current situation must be identified, for comparison to the indices of cases in memory. The CBR system then retrieves the most similar case or set of cases in memory, based on a measure of similarity between the current situation and the stored cases. The process is referred to as *retrieval* and relies heavily upon the *similarity metric* used. Typical similarity metrics include k-nearest neighbor and discrimination nets. Efficient and accurate methods for retrieval and similarity are another major research focus in case-based reasoning.

## Applying Remembered Cases

In applying remembered experiences, the most similar case or cases retrieved by the CBR system are checked for consistency with the current situation. It may be that the retrieved case needs some modification in order to be applied in the current context. For example, in the CHEF system for recipe planning [Hammond 1986], if the current situation calls for duck in place of the meat in an otherwise applicable retrieved recipe, an additional step is required to remove fat from the duck before cooking. This general process is referred to as *adaptation*. If there are portions of the retrieved case that need to be altered, the case is adapted to fit the current situation. The proposed solution can then be evaluated either by directly applying the proposed solution or by some form of model-based or manual critique. Adaptation is one of the most difficult problems being addressed in CBR research.

## The CBR Cycle

Given the results of evaluation, the CBR system may: (1) perform further adaptation and additional evaluation, if the proposed solution both needs to and can be refined; (2) store the results of reasoning as a new case. Case storage represents an incremental learning process, sometimes called lazy learning, that completes the case-based reasoning cycle by committing the new case to memory.

## CBR Knowledge Containers

It is often useful to view a case-based reasoning system in terms of the types of knowledge it employs in reasoning. Richter identified four primary "knowledge containers" that are used in typical CBR systems [Richter 1998]. The *vocabulary* is

used to describe the domain; the *case-base* contains the set of domain experiences; the *similarity measure* is used to retrieve relevant cases; and the *solution transformation* is used in adapting retrieved cases to tailor them for the current situation. These are also represented in Figure 1.1. The knowledge containers are strongly related, and, in principle, the knowledge required to solve a given problem may reside as easily in one as another. A system with complete and fine-grained case knowledge may require very little in the way of adaptation, while a system with powerful adaptation metrics may need very few cases as starting points. It may be appropriate to talk about other knowledge containers as well, for example maintenance knowledge, and the relationships between different knowledge containers are discussed in more detail in chapter 7.

### CBR in Practice

CBR techniques have proven useful in implementing intelligent software components, and CBR systems have been deployed successfully for many types of tasks: for electronic commerce (e.g., [Watson 1997; Stolpmann and Wess 1998; Vollrath *et al.* 1998]); for decision support applications, such as help desks (e.g., [Göker and Roth-Berghofer 1999; Lenz *et al.* 1999]); for planning tasks, such as design and configuration (e.g., [Navinchandra 1988; Sycara and Navinchandra 1989; Stroulia *et al.* 1992; Hennessy and Hinkle 1992; Domeshek *et al.* 1994b]); and for classification tasks, such as diagnosis, prediction, and assessment ([Koton 1989; Bareiss 1989]).

## 1.2 Case-Base Maintenance

CBR systems rely on various knowledge containers, such as the case-base of prior experiences and similarity criteria for comparing situations and retrieving the most relevant cases. Explicit or implicit changes in the reasoning environment, task focus, and user base may influence the fit of the current knowledge state to the task context, which can affect the quality and efficiency of reasoning results. Over time, the knowledge containers may need to be updated in order to maintain or improve performance in response to changes in task or environment. In particular, maintaining the case-base—the traditional mainstay of knowledge underlying CBR systems—is essential for preserving and expanding the capability of a CBR system throughout its life-cycle.

Thus it is important to understand both the essential concepts of maintenance itself, as well as how maintenance practice fits into the design and use of case-based

systems that address task goals. The following sections outline the theoretical issues and practical work in case-base maintenance and beyond that are addressed in this dissertation:

- Case-Base Maintenance Theory
  - Developing a framework for understanding case-base maintenance
  - Investigating regularity assumptions and their relation to maintenance
- Case-Base Maintenance Practice
  - Describing interactive case-base maintenance support in DRAMA, a CBR system to support conceptual aerospace design
  - Describing transformations to support representational maintenance for CBR system construction, the Metamorphoses project
  - Investigating the relationship of performance and competence concerns in guiding maintenance
- Generalizing Case-Base Maintenance
  - Extending case-base maintenance research beyond the case-base
  - Describing similarity maintenance in CBMatrix, a CBR recommender system to support problem-solving in scientific computing

## 1.3 A Framework for Case-Base Maintenance

Experience with the growing number of large-scale and long-term CBR systems has led to increasing recognition of the importance of case-base maintenance in research and practice. Multiple researchers have addressed pieces of the case-base maintenance problem, considering such issues as maintaining consistency (e.g., [Racine and Yang 1997]), preserving competence (e.g., [Smyth and McKenna 1998]), and controlling case-base growth (e.g., [Smyth and Keane 1995]). However, despite the existence of these cases of CBM, there has been little theoretical development of case-base maintenance as a standard aspect of the CBR process. This research defines a general framework of dimensions for characterizing case-base maintenance systems and demonstrates its usefulness in understanding the state of the art in CBM, in unifying current practice, and in suggesting new avenues of exploration by identifying points along the dimensions that have not yet been studied. It helps to solidify the emerging notion in the field of case-base maintenance as a standard aspect of CBR systems, and the framework provides a setting and guide in our development of CBM techniques.

## 1.4 Regularity Assumptions in CBR

Maintenance issues are closely related to fundamental assumptions in CBR. Case-based problem-solving systems reason and learn from experiences, building up case libraries of problems and solutions to guide future reasoning. The expected benefits of this learning process depend on two types of regularity [Leake and Wilson 1999b]: (1) *problem-solution regularity,* the relationship between problem-to-problem and solution-to-solution similarity measures that assures that solutions to similar prior problems are a useful starting point for solving similar current problems, and (2) *problem-distribution regularity,* the relationship between old and new problems that assures that the case library will contain cases similar to the new problems it encounters. Unfortunately, these types of regularity are not assured, and there has been very little prior work toward formalizing these fundamental notions. Even in contexts for which initial regularity is sufficient, problems may arise if a system's users, tasks, or external environment change over time. This research defines criteria for assessing the two types of regularity, discusses how the definitions may be used to assess the need for case-base maintenance, and suggests maintenance approaches for responding to those needs. In particular, it discusses the role of analysis of performance over time in responding to environmental changes.

## 1.5 Interactive Maintenance Support

Over the course of the maintenance work, a number of task-based CBR systems have been developed that address maintenance concerns in service of their task-based reasoning goals. Thus the research makes contributions to the field both in the development of maintenance practice, as well as in developing effective task-based CBR recommendation and support systems. Therefore the maintenance contributions are presented within the overall context of the systems themselves. The first of these is DRAMA, an interactive case-based recommender system, developed to support aerospace design.

Aerospace design is a complex task requiring access to large amounts of specialized information. Consequently, intelligent systems that support and amplify the abilities of human designers by capturing and presenting relevant information can profoundly affect the speed and reliability of design generation. The dissertation describes research on supporting aerospace design in the DRAMA system, which integrates a case-based design support framework with interactive tools for capturing expert design knowledge through "concept mapping" [Novak and Gowin 1984].

By monitoring users as they go about normal high-level design tasks, DRAMA automatically captures user design choices and rationale that can be used to provide proactive recommendations at both the design and design component levels. This helps to maintain the case-base through continuous support for case-authoring and design consistency while significantly ameliorating the knowledge-engineering burden on system users.

In the integrated system, interactive concept mapping tools provide crucial functions for generating and examining design cases and navigating their hierarchical structure, while CBR techniques facilitate retrieval and aid interactive adaptation of designs. The goal of the system is both to provide a useful design aid and to develop general interactive techniques to facilitate case acquisition, adaptation, and maintenance. Experiments illuminate the performance of the system's context-sensitive retrieval during interactive case adaptation and the conditions under which it provides the most benefit.

## 1.6 Representational Maintenance

In building CBR systems for knowledge management and to support corporate memories, it is increasingly important to be flexible in the representation of experience. Achieving widespread case-based reasoning support for corporate memories requires the flexibility to integrate implementations with existing organizational resources and infrastructure. Effective techniques for maintaining case-representations can be extremely useful in deploying case-based systems in many aspects of corporate experience sharing.

Case-based reasoning implementations as currently constructed tend to fall into three broad categories, characterized by implementation constraints: *task-based* (task constraints alone), *enterprise* (integrating databases), and *web-based* (integrating web representations). These implementation types represent the possible targets in constructing corporate memory systems, and it is important to understand the strengths of each, how they are built, and how one may be constructed by transforming another. The Metamorphoses project relates the three types of CBR implementation, discusses their typical strengths and weaknesses, and describes practical strategies for building corporate CBR memories to meet new requirements by transforming and synthesizing existing resources, that is, by appropriate maintenance of the case knowledge representation.

## 1.7 Performance & Competence Relations

An important focus of recent CBR research is on how to develop strategies for achieving compact, competent case-bases, as a way to improve the performance of CBR systems. However, compactness and competence are not always good predictors of performance, especially when problem distributions are non-uniform. Consequently, this dissertation argues for developing methods that tie case-base maintenance more directly to performance concerns. This part of the dissertation begins by examining the relationship between competence and performance, discussing the goals and constraints that should guide addition and deletion of cases. It next illustrates the importance of augmenting competence-based criteria with quantitative performance-based considerations, and proposes a strategy for closely reflecting adaptation performance effects when compressing a case-base. It then presents empirical studies examining the performance tradeoffs of current methods and the benefits of applying fine-grained performance-based criteria to case-base compression, showing that performance-based methods may be especially important for task domains with non-uniform problem distributions.

## 1.8 Beyond the Case Base

It is important to note that the lessons learned from research in case-base maintenance are general, and it is important to take these lessons beyond the case-base. It has been recognized that knowledge containers other than the case-base can be equally important targets for maintenance. Multiple researchers have addressed pieces of this more general maintenance problem as well, considering such issues as how to refine similarity criteria and adaptation knowledge. As with case-base maintenance, a framework of dimensions for characterizing more general maintenance activity, within and across knowledge containers, is desirable to unify and understand the state of the art, as well as to suggest new avenues of exploration by identifying points along the dimensions that have not yet been studied. Here we generalize our theoretical work on case-base maintenance by extending the CBM framework of dimensions to the entire range of CBR knowledge containers. Moreover, we extend the theory to include coordinated, cross-container maintenance. The result is a framework for understanding the general maintenance problem addressed by case-based reasoner maintenance (CBRM). Taking the new framework as a starting point, we explore key issues for future CBRM research.

## 1.9 Similarity Maintenance

An important part of case-based reasoner maintenance is to monitor the effectiveness of the similarity measure. In this dissertation, we present an example of similarity maintenance in the context of CBMatrix, a case-based recommender component developed to support problem-solving in Scientific Computing.

Component-based problem-solving environments (PSEs) provide scientists and engineers with a framework of integrated problem-solving tools and resources that they can easily compose and apply in their particular task domains. Developing effective solution strategies within these environments depends on making good choices about the selection, parameterization, and organization of component tools and resources. Because making good choices may require considerable effort and expertise, designing intelligent components that can make informed recommendations about solution development will play a valuable role in realizing the full potential of PSEs. As part of an overall effort in software component systems and PSEs for scientific computing at Indiana University, the CBMatrix project is developing "intelligent recommender components" that use case-based reasoning methods to assist in selection, organization, and application of scientific PSE tools and resources. The dissertation gives an an overview of the CBMatrix project, the issues involved, and describes the results of experiments in task-based recommendation. In the context of the recommender system, the dissertation describes the similarity maintenance aspect of CBMatrix, which employs genetic algorithms to maintain the similarity knowledge in the case-based recommender component.

## 1.10 Dissertation Overview

This dissertation develops a theoretical foundation for case-base maintenance, uses the theory to help develop practical case-base maintenance techniques and applications, and shows how the case-base maintenance work can be applied more generally to maintaining other knowledge containers. In chapter 2, we present a theoretical framework for describing case-base maintenance techniques according to the types of maintenance policies implemented by a given CBR system. Chapter 3 discusses the regularity assumptions that underly the CBR process, and how variations in these regularities relate to maintenance activity. In chapter 4, we describe the DRAMA tool for aerospace design support, which provides facilities for initial case capture and subsequent refinement that directly exploit user knowledge. Chapter 5 presents a practical model for transforming case-base implementations in the Metamorphoses

project for representational maintenance. Chapter 6 describes new methods for automatically maintaining case-bases by incorporating explicit performance concerns into measures of case-base competence in order to optimize case-base composition. Chapters 7 and 8 illustrate how the work developed for case-base maintenance can generalize across knowledge containers. Chapter 7 extends the case-base maintenance framework to general knowledge container contexts, and Chapter 8 presents an application of similarity maintenance in the context of the CBMatrix case-based recommender system for problem-solving support in Scientific Computing. We conclude in chapter 9 with a discussion of lessons learned, ongoing research, and future directions.

# 2

# Case-Base Maintenance Framework

The growing use of large-scale and long-term case-based reasoning applications has brought with it increased awareness of the importance of maintaining CBR systems. Large-scale CBR systems have become more prevalent, with case library sizes ranging from thousands (e.g., [Cheetham and Graf 1997; Kitano and Shimazu 1996]) to millions of cases [Deangdej *et al.* 1996]. The use of large case-bases raises concerns about the utility problem for case retrieval [Francis and Ram 1993; Smyth and Cunningham 1996], in which the growing cost of case retrieval outweighs the efficiency benefits from additional cases, and has prompted research on controlling case-base growth through compaction policies [Smyth and Keane 1995; Smyth and McKenna 1999a; Zhu and Yang 1999]. Even for smaller case-bases, the difficulties of distributed case collection [Borron *et al.* 1996] and use [Doyle and Cunningham 1999; Watson and Gardingen 1999], as well as the vagaries of real-world data raise concerns about the consistency and accuracy of case knowledge, thus motivating efforts to maintain the case-base to improve its quality [Racine and Yang 1997]. These concerns have led to active research in the area of case-base maintenance (CBM).

Despite a number of projects illuminating these issues for particular CBM systems and tasks, there has been no common framework to guide a more general study of case-base maintenance. Such a framework would be useful for understanding the state of the art in case-base maintenance, illuminating current practice and facilitating the comparison of particular approaches, as has already proven useful for studying case adaptation [Hanney *et al.* 1995; Voß 1996]. A set of dimensions for categorizing case-base maintenance methods can also help to identify problems and opportunities for study, suggesting points of exploration in the space of possible CBM systems. Moreover, a categorization scheme for maintenance approaches is an important step towards cataloging the approaches best suited for particular performance goals.

This chapter presents a framework for describing case-base maintenance. The dimensions developed are used to to characterize CBM policies from CBM systems in the literature. It goes on to discusses the relationship of CBM to the revision of other CBR *knowledge containers* [Richter 1998], and to highlight points for investigation suggested by the framework. It makes no claim of providing a final taxonomy or a complete summary. Nevertheless, the dimensions provide a useful way to describe central aspects of current practice in CBM, and they have led us to identify opportunities for new case-based maintenance approaches. After describing the framework, it sketches how one of these opportunities is explored in CBMatrix, a case-based "intelligent component" [Riesbeck 1996] to assist in using problem-solving environments for Scientific Computing.

## 2.1 Defining Case-Base Maintenance

We define *case-base maintenance* as the process of refining a CBR system's case-base to improve the system's performance:

> Case-base maintenance implements policies for revising the organization or contents (representation, domain content, accounting information, or implementation) of the case-base in order to facilitate future reasoning for a particular set of performance objectives. [Leake and Wilson 1998]

Note that this definition considers the information defining an indexing scheme to be an intrinsic organizational component of the case-base itself. Thus case-base maintenance may involve revising indexing information, links between cases, or other organizational structures and their implementations.

Maintaining case-base contents may affect a single case or multiple cases. It may revise the case representations used (e.g., changing the predicates used to describe domain features); may revise either domain information in the case-base (e.g., correcting an erroneous feature in a case or adding or deleting an entire case) or "accounting" information (e.g., changing information about how frequently a case has been accessed); or may revise how case representations are implemented (e.g., changing from lists to feature-vectors). Thus maintenance of case-base contents may revise the case-base at the implementation level, representation level, or the knowledge level (cf. [Dietterich 1986]).

This definition of maintenance implicitly includes policies for performing CBM indirectly, by revising the maintenance policies themselves. Section 2.4, gives a brief description of one approach to such "meta-maintenance."

## 2.2 CBM Performance Objectives and Constraints

The performance objectives for a CBR system provide criteria for evaluating the internal behavior and task performance of a particular system for a given initial case-base and sequence of problems solved. The choice of case-base maintenance strategies is driven by the maintainer's performance goals for the system and by constraints on the system's design and the task environment. In general, there will be multiple performance measures for a CBR system, and there is no guarantee that all of them can be maximized simultaneously. Smyth and McKenna [1999b] define three types of top-level goals for CBR systems:

1. Problem-solving efficiency goals (e.g., average problem-solving time)
2. Competence goals (the range of target problems solved)
3. Solution quality goals (e.g., the error level in solutions)

These goals may give rise to quantitative maintenance goals (e.g., achieving particular problem-solving time or limits on case-base size), or qualitative ones (e.g., to extend system competence). Smyth [1998] provides compelling arguments for the importance of shaping maintenance policies according to a complete set of performance objectives. Of course, performance objectives may change over time to reflect varying external circumstances, which may necessitate changing (maintaining) maintenance policies as well.

The application of maintenance policies to achieve these goals is also shaped by constraints from the external environment [Leake and Wilson 2000b]:

1. Case-base size limits (if any)
2. Acceptable long-term/short-term performance tradeoffs
3. The expected distribution of future problems
4. The availability of secondary sources of cases

For example, Smyth and Keane's [1995] competence-preserving deletion strategies reflect all of these constraints. Their deletion process keeps the case-base within acceptable size limits (constraint 1); their competence-guided choices are intended to minimize the loss of future coverage (constraint 2); their methods' deletion choices assume a uniform distribution of problems (constraint 3); and no other sources of cases are available for recovering deleted information (constraint 4), making preservation

of competence a key concern. Other instantiations of these constraints would give rise to different strategies. For example, if short-term performance is crucial and long-term is less important, and current problems are concentrated in a small part of the case-base, it may be acceptable to sacrifice current competence and build it back through future learning. Thus a case-based reasoner needs policies for achieving its maintenance goals in light of its constraints. The following section develops a characterization of the properties of case-base maintenance policies.

## 2.3 A Framework for Describing CBM Policies

The goal of a categorization scheme for case-base maintenance is threefold. First, by identifying classes of similar maintenance approaches, such a categorization scheme can shed light on the state of current practice in the field, increasing understanding of current CBM approaches. Second, mapping out the space of candidate approaches helps identify parts of the space that have not been addressed in previous work; these gaps in turn suggest research opportunities. Third, a categorization scheme for maintenance approaches is a first step towards cataloging the most appropriate approaches for particular performance goals.

The framework categorizes case-base maintenance approaches in terms of *case-base maintenance policies* that determine whether, when, and how a CBR system performs case-base maintenance. Maintenance policies are described in terms of how they gather data relevant to maintenance, how they decide when to trigger maintenance, whether they react to problems or proactively forestall them, the types of maintenance operations available, and how selected maintenance operations are executed.

In the framework, *Data collection* gathers, synthesizes, and distills the data about the case base and about system processing; this is the information that will be used to determine whether maintenance operations should be performed. *Triggering* takes this information as input, makes the decision whether maintenance is needed, and selects maintenance actions from a range of possible *Operation types*. *Execution* describes when and how the selected revisions are actually applied to the case-base.

Descriptions generated using the framework characterize basic combinations of policy attributes. A single CBR system may include multiple maintenance policies, each one implementing a different part of the system's overall maintenance agenda (e.g., [Minor and Hanft 2000]). The following dimensions would be used to describe each policy separately. Coordination of maintenance policies is described later in section 7.2.

## Data collection

Data collection gathers information about individual cases, about the case base in part or as a whole, and/or about the overall processing behavior of the CBR system. Data collection about individual cases might record the number of times a case has been successfully used or the number of times it has failed. Data collection about the case base as a whole could involve, for example, monitoring the size of the case base. Data collection about processing might involve noting clusters in input problems, input problems that the system is unable to solve successfully, or input problems for which processing costs are too high.

*Type of data: None, Synchronic, or Diachronic.* There are three approaches to collecting and analyzing data to decide when case base maintenance is needed. The simplest is to do no collection at all. A policy with no data collection makes maintenance decisions independently of the present or past state of the case base. As such, this type of policy is referred to as *non-introspective*. For example, a CBR system that updates its case-base by unconditionally adding a case each time it adapts a prior case would need no data collection. This is the approach of most CBR systems. Similarly, a system may drive maintenance according to external information sources. This is valuable for proactive maintenance, for example, to add cases to a help desk case-base in anticipation of future queries.

More sophisticated reasoning is enabled by considering a snapshot of the current case-base in part or as a whole. Examination of this information can determine, for example, whether a case is worth adding to a case-base because it increases the competence of the CBR system, or whether a solution can be discarded without affecting competence [Smyth and Keane 1995]. As another example, Reinartz et al. [2000] propose a set of measures that can be computed to assess the overall quality of a case-base in order to trigger maintenance. Policies that consider snapshot information are called *synchronic*.

The most informative approach is to collect data over time, over a sequence of snapshots, in order to identify trends in how case-base contents and usage are changing. Policies that consider changes in the case-base over time are called *diachronic*. For example, a policy that gathered information about trends in retrieval times, to identify the onset of utility problems, would be diachronic. Note that even though information may be accumulated over time (e.g., the number of times a case is used successfully), the element of time must be an intrinsic part of the picture in order to constitute a diachronic policy (e.g., frequency of a case's usage, increasing or decreasing). Because synchronic and diachronic collection examine the internal state of the case-base, both are referred to as *introspective*.

*Timing: Periodic, Conditional, or Ad Hoc.* A maintenance policy must specify when data collection is performed. In our framework, there are three possibilities. *Periodic* timing happens at a set frequency with respect to the CBR cycle. For example, data collection might be performed after each problem-solving cycle. Periodic timing that happens every cycle is termed *continuous*. *Conditional* data collection is performed in response to a well-defined but non-periodic condition. For example, analysis might be triggered whenever the number of cases in the case library reaches a particular threshold [Smyth and Keane 1995]. *Ad hoc* timing happens under ill-defined conditions determined externally to the CBR system[1]. Examples of ad hoc timing are user-initiated tests on the case base to determine whether maintenance is needed or a domain expert's decision to add new cases regardless of the case base contents.

*Integration: On-line or Off-line.* Data collection may operate *on-line*, during the course of an active reasoning episode, or *off-line*, during a pause in reasoning, such as waiting for user input or when idle between reasoning episodes. The choice between on-line and off-line processing may affect the resources that can be devoted to the analysis process, making it important for determining whether a policy is appropriate for time-constrained processing.

## Triggering

The results of data analysis serve as input for determining whether case-base maintenance is necessary. Both the *timing* and *integration* dimensions discussed previously apply to this step as well. Maintenance triggering evaluates whether to perform maintenance, selects maintenance actions to use, and may set parameters to guide their future execution (e.g., determining when they will be performed). Triggering can be done periodically, conditionally, or on an ad hoc basis, and on-line or off-line.

Conditional triggering can be subdivided into three classes depending on the conditions that determine whether maintenance is triggered: *space-based* (e.g., filling a limited amount of case storage), *time-based* (e.g., retrieval time exceeding a threshold), or *result-based* (e.g., the system failing to solve a given problem or the wrong case being retrieved).

[1]This category name in no way implies that the choice is ill-considered; simply that it is not under control of the policy.

## Proactive vs. Reactive Maintenance

Case-base maintenance is often seen as a process of detecting problems and responding to repair them (e.g., for case inconsistencies or exceeding case-base size limits). In that case, maintenance is triggered by conditions typically indicative of system failures. However, maintenance may also be proactive, taking steps despite successful performance, to avoid predicted future problems or improving future performance. For example, a software company with a case-based help desk system might perform maintenance in advance of the launch of a new product, in order to seed the case-base with cases expected to be useful after the product has been released.

## Operation types

Different maintenance policies revise different types of information (the *target type*) at different levels (the *revision level*).

*Target type.* For case-base maintenance, revision operations can focus on three types of targets: *Indexing structures*, *domain contents*, and *accounting information.* As will be described in section 2.4, *maintenance policies* themselves can also be targets for CBM.

*Revision level.* Revision operations can make revisions with ramifications at three levels: Affecting only the *implementation level* (e.g., changing an indexing structure from a list to a D-tree when the case-base exceeds a certain size or changing case representations from lists to vectors), affecting the *representation level* (e.g., reconciling inconsistent feature names or case formats in cases that come from different sources), or affecting the *knowledge level* as well (e.g., correcting an erroneous feature value, generalizing case values, or adding or deleting cases).

Finer-grained characterizations of operator types are of course possible (e.g., Heister and Wilke [1998] and Reinartz et al. [2000] describe sets of atomic maintenance operations). However, as with the rest of the categorization scheme, we have used higher-level categories to facilitate cross-system comparisons of major characteristics.

*Scope of Maintenance: Broad or Narrow.* A given operation may be applied locally, to few items in the case base, or more globally. Operations that affect a single case or a small subset of the case-base have *narrow* scope, and operations that affect a large subset or the entirety of the case base have *broad* scope. This dimension is especially useful when characterizing resource-bounded processing.

## Execution

Execution is characterized by the timing of maintenance operations and their integration with other system processing. Execution timing is described using the timing dimension previously described for data collection (periodic, conditional, or ad hoc); timing may also be "none" for systems with no execution. For example, a maintenance policy may simply inform a maintainer that maintenance is needed without making changes (none); changes may be made on a regular basis (periodic); changes may be held for batch updating when enough cases are accumulated (conditional); or changes may be held for when an expert is available (ad hoc). Likewise, execution integration is described as on-line or off-line depending on whether maintenance operations are performed during or between reasoning episodes.

## Categorizing Policies for Case-Base Maintenance

To illustrate the use of the framework and to understand the range of CBM methods, the framework is applied to a sampling of CBM approaches, beginning with a few simple examples. In describing particular maintenance policies, two parts of the CBM framework that are particularly useful for describing current CBM systems should be emphasized: the type of data collected and how maintenance policies are executed. Table 2.1 summarizes the described approaches along these dimensions.

## Policies targeting domain content

Policies targeting domain content may be divided into policies aimed at adding and deleting cases, and policies aimed at revising internal case content. We first consider addition and deletion policies, and then policies to refine the cases themselves.

*Standard case learning and manual maintenance:.* The standard learning of CBR problem-solving systems (always adding each new case to the case base) is designated in the table as $CBR_1$. No data analysis is performed—the new case is recorded without considering the existing contents of the case base—so it is non-introspective. Because learning happens during each reasoning cycle, this policy is continuous (periodic) and on-line. Because only a single case is added, the scope of change is narrow.

Another common CBR method ($CBR_2$) involves a non-learning system maintained by a domain expert who sometimes adds a variable number of new cases. For this method, we presume no system analysis of the existing case-base, so the maintenance policy is non-introspective. Because the timing of the updates depends on the expert's

| | | | | Data Collection: *Type of Data* | |
|---|---|---|---|---|---|
| *Activation Timing* | *Integration Type* | *Scope of Changes* | None | Synchronic | Diachronic |
| Execution: Periodic | On-line | Broad | | | |
| | | Narrow | $CBR_1$ | Muñoz-Avila | |
| | Off-line | Broad | | | |
| | | Narrow | | | |
| Conditional | On-line | Broad | | Fox & Leake | |
| | | Narrow | Leake & $Wilson_2$ | Smyth & $Keane_2$; Surma & Tyburcy; Hammond; Ihrig & Kambhampati | Leake & $Wilson_1$ |
| | Off-line | Broad | | Smyth & $Keane_1$; Portinale et al. | |
| | | Narrow | | | |
| Ad hoc | On-line | Broad | | | |
| | | Narrow | | $Minor\&Haft_1$ | |
| | Off-line | Broad | $CBR_2$ | Aha & Breslow; $IBL_n$; Watson; Racine & $Yang_{1,2,3}$; Netten; Smyth & $McKenna_{1,2}$; Zhu & Yang; Göker&Roth-Berghofer; Yang & Wu | |
| | | Narrow | $CBR_2$ | Watson | |
| No Execution | | | | Shimazu & Takishima | |
| | | | *Non-Introspective* | *Introspective* | |

Table 2.1: Sample CBM approaches placed along major dimensions

external decision, the timing is ad hoc. Because the cases are added manually outside of normal processing, the integration is off-line. Because the number of cases can be small or large, the scope varies from narrow to broad.

*Additional policies aimed at case retention:.* Smyth and Keane [1995] describe a competence-preserving approach to case deletion, which specifies a case utility hierarchy in terms of coverage and reachability. When the number of cases in the case-base exceeds the "swamping limit," their "footprint-utility deletion" strategy selects candidates for deletion based on the utility hierarchy. Because the hierarchy is defined with respect to the current state of the case-base, the policy is synchronic. Because maintenance is triggered in response to the current size of the case-base, timing is conditional. Smyth and Keane describe this mechanism as being applied either to small numbers of cases during processing, using a heuristic method of utility evaluation (Smyth & $Keane_2$, on-line and narrow) or to large numbers of cases with full analysis outside of the reasoning cycle (Smyth & $Keane_1$, off-line and broad).

Surma and Tyburcy [1998] describe policies for replacing older cases as new cases are learned, in order to bound case-base size to maintain bounded retrieval time.

The policies act on the current state of the case-base (synchronic) during the storage phase (on-line) when the size limit is reached (conditional) to make a narrow change.

Smyth and McKenna [1999a] present a policy for case-base editing/compaction (Smyth & McKenna$_1$) that uses an explicit case competence model based on notions of coverage and reachability. Their "relative coverage" metric provides a precise measure of competence contributions for individual cases. This allows the case set to be ordered by likely competence contribution. To build the case base, the ordered set is presented to a condensed nearest-neighbor algorithm that successively retains only those cases that are not solved by a case that has already been retained. This method examines the current state of the case-base (synchronic). It is presented as a way to edit the entire case-base during construction (ad hoc, off-line, broad), though they have also developed efficient methods for keeping the reachability and coverage measures current as the system is used [Smyth and McKenna 2000].

Muñoz-Avila [1999] presents a case retention policy based on retrieval benefits to case-based planning. After a problem-solving episode, adaptation effort is analyzed to determine whether the guidance of retrieved cases was "beneficial" (the new case need not be stored) or "detrimental" (the new case is added). This policy is synchronic, periodic, on-line (pre-storage), and narrow.

Portinale, Torasso, and Tavano [1999] present a strategy for managing case memory by removing "useless" cases (that have not been retrieved before an expiration limit) and "false positive" cases (that have been retrieved and have had more adaptation failures than successes). Memory management is conditionally triggered after a variable length time window that is tailored to the growth of case memory and reasoning failure rate. This policy uses synchronic information, conditional timing, and off-line integration to make broad changes.

Zhu and Yang [1999] describe a case-addition algorithm for case-base compaction that uses a problem-neighborhood model of case coverage. Cases are successively added based on added benefit/usefulness to the neighborhood of the case-set retained so far. The analysis is synchronic, with ad hoc timing, off-line integration, and broad scope.

*Policies aimed at internal case content:.* A number of proposed CBM policies are aimed at internal case content. Shimazu and Takashima describe a version of the CARET system that identifies discontinuities in a case-base [Shimazu and Takashima 1996]. That system uses synchronic data collection; it retrieves a set of "Maybe Similar Cases" (MSCs), chooses a single best "Base Case" (BC), and classifies as "discontinuous" any remaining MSCs whose suggestions differ from the BC by more than a given threshold, identifying them as potential candidates for maintenance. However, the system does not execute revisions, so the policy has no execution.

Racine and Yang [1997] describe policies for identifying redundant cases (Racine & Yang$_1$) and inconsistent cases (Racine & Yang$_2$). Both policies rely on an analysis of the current state of the case-base, so they are synchronic. Both are applied to the case-base as a whole when desired by a case-base maintainer, so they are broad, ad hoc and off-line.

Minor and Hanft [2000] describe a framework to support interactive revision of case content over case "life-cycles" (Minor & Hanft$_1$). Support for revising case-content is based on the current state of the case-base and is interactive (ad hoc and on-line), with narrow changes being made to individual cases.

Leake and Wilson [1998] describe a maintenance policy that updates case contents with a revision policy installed in response to trends in performance anomalies (Leake & Wilson$_2$), enabling a lazy update of the case-base. The installed policy always checks (no analysis) whether a retrieved case (on-line between retrieval and application) has been updated to reflect a previously detected trend (conditional timing), and updates just that case (narrow scope) in situ before passing it on for further reasoning.

Additional approaches address both the presence of cases in the case base and their internal content. Watson [1997] presents a set of guidelines for human case-base maintainers that involve performing periodic tests on the entire case-base. This policy can be described as having synchronic analysis, ad-hoc timing, off-line execution, and narrow or broad scope.

Netten [1999] presents a framework for verification of case-base integrity in case-based diagnosis systems. This includes checks for redundancy, inconsistency, and incompleteness in case definitions, as well as verification of coverage, reachability, and accuracy. The framework is presented as a means to validate a diagnosis case-base before being applied in critical environments. It makes use of the current state of the case-base, is ad hoc, off-line, and broad in scope.

Göker and Roth-Berghofer [1999] describe the "Maintenance Cycle" of the HOMER case-based help-desk support system, which includes a policy for verifying whether new cases entered by help-desk operators should be added to the central case repository. Redundancy and inconsistency are checked by a case-base administrator. Potential cases are checked against the current case-base (synchronic), at a time determined by the maintainer (ad hoc), separate from ongoing processing (off-line), and only for cases under consideration (narrow).

## Policies targeting indices

A number of classification systems using Instance-Based Learning (IBL) and related techniques ($IBL_n$) include policies for eliminating noisy and redundant instances from a set of training examples (cases). These systems generalize a case-base either explicitly, by merging cases with similar coverage (e.g., [Domingos 1995]) or implicitly, by choosing a smaller, representative subset of cases (e.g., [Aha *et al.* 1991]). Such policies typically consider a static set of cases (synchronic), are user-initiated (ad hoc), perform execution off-line, and are applied to the entire training set (broad). Because case features (other than the category) are only used as indices, we view their generalizations as revising indexing information. When IBL systems remove noisy instances or remove a class entirely, their target is domain content.

Many methods have been proposed for selecting case indices. Some are included in the standard case addition process ($CBR_1$), as in the model-based approach of Bhatta and Goel [1995]. Others, however, adjust current indices in response to performance deficiencies. Hammond [1989] describes a failure-driven method for explanation-based selection of new indexing features. Likewise, Ihrig and Kambhampati [1997] describe a policy that explains plan replay failures in order to add features to check during future retrievals. These policies are conditional, on-line, and make narrow changes.

Fox and Leake [1995a] describe a policy that triggers index revision for plan cases in response to plan failures. This policy considers snapshot information about execution (synchronic), is executed conditionally, is performed on-line, and revises indices in the entire case-base (broad scope).

Aha and Breslow [1997] describe an index revision method for conversational CBR that considers an entire case-base to optimize interactive question paths in response to an external request. This policy has synchronic data collection, ad hoc activation timing, off-line integration, and broad scope.

Racine and Yang [1997] describe a policy for deriving and updating indices of unstructured cases (Racine & $Yang_3$), using methods derived from information retrieval. Like their other policies, this policy is synchronic, broad, off-line, and has ad-hoc execution.

Smyth and McKenna [1999b] present a method for index refinement (Smyth & $McKenna_2$) based on competence groups defined with reference to measures of coverage and reachability. From each competence group, a set of footprint cases that cover the remainder of the group are used to focus retrievals. The indexing organization uses synchronic information, ad-hoc and off-line, to make broad changes.

Yang and Wu [2000] describe a method in which a large original case base is partitioned into a distributed set of case-base clusters using density-based clustering

methods. Retrievals are made from the distributed case organization by finding the best case cluster, then the best case within that cluster. The clustering method uses a broad snapshot of the original case-base, and the smaller case-bases are described as being built by an expert based on the clustering result, ad hoc and off-line.

### Policies targeting maintenance policies

Leake and Wilson [1998] describe a diachronic maintenance policy (Leake & Wilson$_1$) that detects potentially important trends in performance anomalies on-line, based on the conditional strength of the trend, and responds by installing a new maintenance policy tailored in response to the trend detected. It performs a narrow change—adding a new maintenance policy. This type of policy is described in more detail in Section 2.4.

## 2.4 Meta-Maintenance by Lazy CBM

When a CBR system retrieves a case and adapts it to fit a new situation, CBM normally stores the result of adaptation as a new case and leaves the original case unchanged. However, if there are defects in the old case, case-base maintenance can simultaneously revise the old case and re-store it in its updated form. This approach updates old cases in a "lazy" manner as they are applied to new situations. It is driven by a process similar to case adaptation, but whose aim is to repair a problem in an old case rather than to fit that case to a specific new situation. This allows expensive updates to be performed only on the portion of the case base that is actually being used, decreasing update effort while still allowing future processing of frequently-used cases to start from the updated versions. Thus when a change must be applied throughout the case-base, a CBM system can either (1) make that change to all old cases simultaneously, when it next performs overall maintenance, or (2) generate a maintenance rule to update each case that is retrieved, when it is retrieved (and before it is applied to the new situation). Installation or revision of these maintenance rules can be viewed as a form of "meta-maintenance," maintaining the system's maintenance knowledge. Note that both approaches achieve a broad change, but that the second does so by implementing two policies with narrow scope, making it preferable in time constrained circumstances.

With a lazy updating scheme, cases that are obsolete must be distinguished from cases that have been updated. When a sufficiently large proportion of the retrieved cases have already been modified by a maintenance rule, it may be possible to abandon

the rule—the case base may have been sufficiently modified for the problems the system tends to encounter.

In rapidly-changing domains, especially if application of maintenance rules is inexpensive, it may be preferable never to update the stored cases, instead composing old maintenance rules into new ones to obtain the desired net changes (e.g., to take compound inflation into account). Such a method also facilitates retraction of invalid updates: flawed maintenance rules can be retracted without making any changes to existing cases.

## 2.5 Trends and Lazy Maintenance

To illustrate how systems can respond to detected trends, an example is drawn from our work in CBR for Scientific Computing. In scientific computing, problem solving environments (PSEs) provide scientists with a framework of integrated problem-solving tools that they can easily configure and apply to problems that arise in their particular task domains. Because effective solution strategies depend on making good choices about the organization and configuration of these tools, considerable expertise may be needed to achieve full benefit from the tools provided by a PSE. However, it is often difficult to capture principles guiding tool and parameter selection. Consequently, CBR methods to guide tool selection, organization, and application have the potential to play a valuable role in PSEs. The CBMatrix project investigates CBR and CBM issues arising in the context of CBR components within a scientific PSE, the Linear System Analyzer (LSA) [Gannon *et al.* 1998], which is aimed at aiding the solution of sparse linear systems. Given a scientific computing problem to solve within the LSA, CBMatrix retrieves prior cases that suggest computational methods and parameters for solving the problem efficiently (e.g., the data structures to use to achieve the highest megaflop performance rating). We discuss the CBMatrix project in more detail in chapter 8.

The PSE advisory task requires the management of substantial case libraries in the face of unreliable information, limited feedback, limited storage, and changing external circumstances. A particularly acute issue concerns how to revise the case-base to improve performance when classes of problems change (e.g., when a scientist begins to apply the scientific computing system to a series of problems with different characteristics from those for which the case base was built) or when changes in the external environment affect the quality of the advice offered by a pre-existing case-base (e.g., if the scientist runs CBMatrix on one set of problems, on one computer, to build a library of advice on methods for solving those problems, and then buys a new computer with hardware that renders some of the prior advice obsolete). Thus this

domain requires addressing not only the maintenance issues involved in dealing with potentially noisy and unreliable data (e.g., because results depend on the external load on the machine), but also on addressing questions about how to maintain a case-base when new hardware requires systematic changes in the recommendations the CBR system provides.

The CBMatrix system implements two maintenance policies that together result in a lazy update of the case-base by a "pre-adaptation" revision of retrieved cases. The first policy installs a new maintenance rule when needed, as described in the previous section. This policy is triggered by diachronic analysis of successive snapshots of the case-base as new situations are processed, in order to recognize changes in machine characteristics. The data collection process for this policy monitors the predictions made by retrieved cases about the expected performance of the most appropriate data structure for solving a given system. If the processing results in performance that is either significantly worse (unexpected failure) or significantly better (unexpected success), the result is added to a data set that is analyzed for trends in performance. Individual fluctuations might be due to processing loads, etc., while consistent trends suggest a more durable change.

The number and magnitude of the unexpected successes or failures with respect to time (measured in numbers of reasoning episodes/cycles) define a trend in performance anomalies that can indicate a changing trend in the linear system processing results (e.g., because the computer being used to solve the problems has been upgraded). Once a trend has achieved a certain level of activation, this maintenance policy installs the second maintenance policy, a new maintenance rule to adjust subsequent predictions (e.g., if there were a trend for predictions to be 20% pessimistic, the rule would adjust predictions upwards on each retrieved case that had not yet been adjusted). This is a simple approach to a problem that is in general very complex, but it appears practical for this type of change and shows the benefit of considering diachronic information when triggering maintenance.

## 2.6 Maintenance and Overlapping Knowledge Containers

The multiple knowledge containers of CBR overlap; knowledge available in one can replace missing knowledge in another [Richter 1998]. Likewise, the effects of maintenance to one knowledge source may be equivalent to maintenance on another. For example, the same overall effects on system accuracy might be achieved by case-base reorganization—which we consider part of case-base maintenance—or by adjustment

of the similarity measure—which we consider external to CBM. Although this framework focuses only on case-base maintenance, in general CBM can be viewed as part of the larger task of CBR system maintenance (e.g., [Heister and Wilke 1998]), and we discuss issues of maintenance in the large in chapter 7.

Given the context of the framework for describing maintenance policies, the next chapter describes how fundamental regularity assumptions in case-based reasoning can affect the implementation of case-base maintenance strategies.

# 3

# Regularity

Case-based reasoning solves new problems by retrieving stored cases encapsulating records of similar problems, and adapting their lessons to fit the new circumstances. Case-based problem-solving is based on two central premises about the regularity of the problem-solver's world [Leake and Wilson 1999b; Kolodner 1993]. The first, which we will call *problem-solution regularity,* describes the relationship between problem descriptions and solutions that assures that similar problems have similar solutions. This regularity is needed to guarantee that cases for similar prior problems are likely to be useful starting points for new reasoning. The second, which we will call *problem-distribution regularity,* describes the relationship between new problems and those previously encountered. This regularity is needed to assure that the system will have the cases it needs for the problems it is called upon to solve. The relative strength of these assumptions as embodied in particular CBR systems over time can provide a strong indication of the need to maintain the case-base.

The successes of numerous CBR systems bear out that for many tasks and domains, appropriate similarity metrics can be devised to provide sufficient problem-solution regularity, and that problem-distribution regularity is often sufficient to enable effective CBR. Unfortunately, no matter how good initial similarity metrics might be for a given task and domain, and no matter how complete a case library a system may build up, changes in task and domain characteristics may render obsolete prior similarity criteria or cases. Developers have cited the problem of dealing with changing task characteristics as the reason for rejecting CBR for some tasks [Talebzadeh *et al.* 1995], and the long-term use of CBR systems makes such changes increasingly likely during a system's lifetime. In order to perform as well as possible despite changing circumstances, a CBR system must be able to evaluate how well the regularity assumptions apply and to signal the need for maintenance or to invoke its own maintenance strategies as needed.

This chapter presents steps towards understanding and responding to deviations from desired regularities as part of maintenance. First, it defines measures that can be used to calculate the amount of problem-solution regularity and problem-distribution regularity that exist for the problem sequences that a system encounters. Second, it discusses methods that may be used to respond to, and (ideally) to exploit changing characteristics of the problems the CBR system solves and of the environment in which its solutions must be applied.

In particular, it describes opportunities for maintenance strategies that perform their changes based on analysis of problem-solving and case-base characteristics over time—*diachronic case-base maintenance strategies* as described in [Leake and Wilson 1998]. In general, determining the right response to shifting context requires knowledge that is unlikely to be available from a single snapshot of the CBR system's state. However, by examining *trends* in retrieval performance, system errors, and presented problems, the system may be able to respond more effectively.

## 3.1 Defining Regularities for CBR

It is well-known in the CBR community that case-based reasoning depends on two relationships: the relationship between *similarity of problems* and *similarity of solutions*, and the relationship between *prior problems* (solved by the system or provided as seed cases) and *new problems*. However, to our knowledge, there were not yet precise definitions of what these relationships mean, prior to the introduction of this work in [Leake and Wilson 1999b]. Such definitions are useful to quantify and compare the relationships in order to understand the effects of different similarity metrics, case bases, and problem sequences on the performance of different CBR systems. Equally important, such definitions give criteria for monitoring the appropriateness of a system's similarity criteria and case library for dealing with current problems, in order to identify the need for system maintenance. This section proposes working definitions as a basis for discussion and study.

### Basic Assumptions and Definitions

The regularity definitions make some standard assumptions throughout. First, it is assumed that there is a fixed CBR system that processes problems in a problem space $P$ and that the solutions for these problems are elements of a solution space $S$. Cases are pairs $(p, s) \in C = P \times S$, the set of all possible cases. The system begins with a finite "seed" case base $B_1 \subseteq C$. As the system is used, it processes a sequence

of problems Q = $p_i$, $p_{i+1}$, ... , $p_j$, where each $p_k \in P$ for $k = i, ..., j$. The sequence is defined to start with an arbitrary index because, as discussed in section 3.7, it is sometimes useful to consider the subsequence that starts after some initial set of problems has been processed.

**Adding to the case base:** We assume that after each problem is processed and the resulting solution has been evaluated, a new case with the problem and its correct solution are added to the case base. This means that each problem $p_k$ is processed using an updated case base $B_k$ that includes the results of previous processing. Note that this does not imply that the system can solve all problems presented to it: The correct stored solution may be based on external feedback if the system generates an incorrect solution or fails to generate a solution.

**How problem distance guides retrieval:** The CBR system uses a "problem distance" function $PDist : P \times C \rightarrow [0, \infty)$ to measure the distance between a new problem and the problem description of a stored case. $PDist(p, c)$ is zero if $p$ is the same problem solved by $c$. Given a new problem, the CBR system retrieves the case closest to that problem according to $PDist$. However, there is no guarantee that the case considered closest by this function will actually be "close" to the problem in any useful way. This function simply reflects the similarity metric built into the system, whether or not it is useful.

**How usefulness of retrievals is judged:** The evaluator of the system uses a "real distance" function $RDist : P \times C \rightarrow [0, \infty)$ to measure how far the solution in a case is from the solution for a given problem. This function measures the usefulness of retrieved solutions according to the evaluator's goals for the retrieval process, which may not be classic "similarity." For example, if the evaluator's primary goal is to minimize the adaptation time required to generate a new solution, "real distance" could be measured in adaptation time: $RDist(p, c)$ could be the time to adapt the solution from case $c$ to solve problem $p$, with some upper limit on the amount of time allowed. $RDist$ could also be defined to reflect other retrieval goals. For example, if reliability of adaptation is an issue, it could consider cases "closer" to a problem if they can be adapted to solve the problem using more reliable adaptations (regardless of adaptation time). Likewise, for case-based planning, if execution cost is an issue $RDist$ could consider cases closer if they yield solutions that can be executed at lower cost.

It should be emphasized that $RDist$ does not necessarily correspond to any function within the CBR system; it is an external criterion. For example, $RDist$ might

be calculated off-line to determine the retrievals the CBR system *should have* made. Thus efficiency of calculating the $RDist$ function is comparatively unimportant. It might be possible, for example, to calculate $RDist$ for adaptability by simply adapting all stored cases to the new problem and seeing which adaptation was fastest.

In an ideal CBR system, the cases with the closest *problems* (according to $PDist$) would also have the closest *solutions* (according to $RDist$). In practice, of course, the actual similarity metric is likely to differ from the ideal (see [Smyth and Keane 1996], for an empirical demonstration). In some situations the deviations may be substantial enough to impair system performance.

## Defining Problem-Solution Regularity

The goal of this definition of problem-solution regularity is to capture how well $PDist$ approximates $RDist$ in practice. Because this depends on the specific context in which the CBR system is solving problems, this definition explicitly depends on:

- the goals for retrieval (as captured by $RDist$),
- the set of seed cases available to the system, and
- the problem sequence that the system is called upon to solve.

As background for the definition, for any input problem, we can calculate two sets of cases according to the formulas below. The first set of cases, designated by CCP for *Closest Cases to Problem*, contains all the cases within a case base B whose problem descriptions are closest to the input problem. The second, designated by RCC for *Real Closest Cases*, contains the cases whose solutions are within a user-specified neighborhood of the optimal solution. The size of the neighborhood is determined by a user-specified non-negative parameter $\epsilon$.

$$CCP(PDist, p, B) = \{c \in B | PDist(p, c) = min_{c' \in B} PDist(p, c')\} \quad (3.1)$$

$$RCC(RDist, p, B, \epsilon) = \{c \in B | RDist(p, c) \leq min_{c' \in B} RDist(p, c') + \epsilon\} \quad (3.2)$$

If $\epsilon = 0$, $RCC$ returns the optimal cases for solving the problem according to the "real" distance metric.

We let $B_k$ designate the case library used when processing problem $p_k$. This case library contains the initial seed cases and all the new cases added to the case base

processing problems before $p_k$. Following the notion of *precision* in information retrieval, we then define:

$$SimPrecision(PDist, RDist, p_k, B_k, \epsilon) = \frac{CCP(PDist, p_k, B_k) \cap RCC(RDist, p_k, B_k, \epsilon)}{CCP(PDist, p_k, B_k)} \tag{3.3}$$

This function measures the probability that a case returned as optimal by the similarity function will actually be within $\epsilon$ of an optimal case.[1]

Given these definitions, problem-solution regularity is defined as the average value of $SimPrecision$ over the problem sequence $Q$, starting with case base $B_i$, as follows:

$$ProbSolnReg(PDist, RDist, Q, B_i, \epsilon) = \frac{\Sigma_{k=i,\ldots,j} SimPrecision(PDist, RDist, p_k, B_k, \epsilon)}{j - i + 1} \tag{3.4}$$

When $\epsilon$ is set to 0, this function calculates the average probability that a case for a maximally-similar problem will actually be optimal. With non-zero values for $\epsilon$, this function provides information about the average probability that a maximally-similar problem (according to the system's similarity metric) will be acceptably close to a maximally useful case, which determines the quality of the similarity metric.

Note that when $ProbSolnReg$ is used to compare the problem-solution regularity of different systems, $RDist$ must be same for both systems. If different systems have different "real" costs (e.g., because of differences in adaptation capabilities), differences in the values of $ProbSolnReg$ for the two systems may not predict their relative performances.

## Defining Problem-Distribution Regularity

The second regularity assumption of CBR is that new problems will tend to resemble the problems addressed in previous cases (either in the seed case base, or in cases learned during prior processing). This is referred to as *problem-distribution regularity*. It determines the likelihood that, as new problems are processed (and new cases with their solutions are added to the seed case base), the case base will contain

[1] Because we assume that the system will reason from a single most similar case, the IR notion of *recall* is not relevant here. It would be relevant if, e.g., the system attempted to increase reliability by generating and comparing solutions starting from multiple cases.

cases for similar problems. When the case base does contain similar problems, and when (in addition) there is sufficient problem-solution regularity, this will result in retrieval of cases whose solutions are close to the actual solutions according to $RDist$.

$ProbDistReg$ calculates the percentage of cases in a problem sequence $Q = p_i, ..., p_j$ for which there are sufficiently close cases in the current case bases $B_k$ built up from the seed case base $B_i$, according to a user-specified distance limit $\epsilon \geq 0$.

$$ProbDistReg(Q, B_i, \epsilon) = \frac{1}{j-i+1} * \Sigma_{k=i,...,j} \begin{cases} 1, & \text{If } min_{c \in B_k} PDist(p_k, c) < \epsilon \\ 0, & \text{Otherwise} \end{cases} \quad (3.5)$$

Together, $ProbSolnReg$ and $ProbDistReg$ provide measures that describe the performance of a CBR system. Individually, each one identifies problems that can be addressed by either refining the similarity metric or the solutions stored in cases (for $ProbSolnReg$) or by adding to the case library (for $ProbDistReg$).

## 3.2 Perspective on Regularity-Related Research

This section considers the importance of the regularities and compares the perspective here to related research; the following sections examine its practical application.

*Work on Problem-Solution Regularity:.* The importance of problem-solution regularity underlies the considerable attention to similarity criteria in CBR research. Faltings [1997] uses probability theory to prove that for prediction tasks, the assumption that a problem with similar features to an earlier one is likely to have a similar solution is guaranteed to be true on average. The issue of how to define practical similarity metrics for particular tasks remains a central research focus of the field, making it useful to have criteria for comparing different similarity metrics.

Recent CBR work has developed methods for making retrieval criteria explicitly reflect the underlying "true" retrieval criterion that has been called $RDist$. A primary example is adaptation-guided retrieval [Smyth and Keane 1996], which replaces the traditional similarity criterion with estimated cost of adaptation, in order to retrieve cases that satisfy the goal of easy adaptation.

*Work on Problem-Distribution Regularity:.* The key question of problem- distribution regularity is whether the case library will contain the cases a system needs to solve the problems it encounters. The importance of problem-distribution regularity

is recognized by developers of CBR applications, who attempt to gather representative and well-distributed sets of cases for their systems (e.g., [Kriegsman and Barletta 1993; Watson 1997]).

Recent work on case-base competence [Smyth and McKenna 1998; Zhu and Yang 1999] has developed methods for estimating the range of problems that can be solved by a system with a given case-base. The purpose of this work is to assure that problem-solution regularity is sufficient, to give an indication of the likely system success rate, and to help identify regions of the case base in which additional cases may be needed.

Problem-distribution regularity is closely related to case-base competence, but this work differs in two ways. The first difference concerns the role of problem distribution. Analysis of case-base competence assumes a uniform distribution of problems in order to make analysis more tractable. Likewise, it is customary for empirical evaluations of CBR systems to use a randomly-generated set of problems uniformly distributed in the problem space (e.g., [Veloso 1994]). However, our definition explicitly references the particular problem sequence on which the behavior is measured. While we agree with Smyth and McKenna [1998] that assuming a uniform distribution can provide a very useful overall view, considering specific details of problem presentation order and distribution can be useful as well. For example, the quality of a CBR system's performance can depend strongly on the order of case presentation [Fox 1995; Redmond 1992], making it desirable for the formulas to be usable for exploring the effects of different orderings. Likewise, as discussed later, if the system can identify "hot spots" in case-base accesses, examining problem distribution regularity may make it possible to reorganize the case base to speed likely retrievals, or to delete (or deactivate, e.g., by placing in secondary storage) cases that are not being used.

Second, the definition of problem-distribution regularity depends on a user-defined threshold for what constitute sufficiently similar stored cases, rather than considering only whether the problem can or cannot be solved. Using a user-defined criterion for whether a stored case is "close enough," rather than simply whether *some* solution can be generated, is important when the quality of solutions depends on the amount of adaptation performed, or when there are changeable limits on the amount of effort that can be expended on adaptations. For example, in some domains, available domain theories are strong enough for local adaptations but are not sufficiently reliable for more substantial changes (e.g., [Cheetham and Graf 1997]).

*Work on Case-Base Maintenance:.* Case-base maintenance research addresses issues such as assuring that the cases in the case base cover the space of possible problems [Smyth and McKenna 1998; Zhu and Yang 1999] and deleting superfluous cases to improve space efficiency or utility of retrieval [Smyth and Keane 1995].

These do not address, however, how to maintain the case-base in response to specific task needs—for example, to build coverage in precisely those areas that tend to arise in current problems—or how to predict the need for future maintenance from current problems, in order to proactively revise the case-base before problems occur. Salganicoff [1997] has studied the problem of learning time-varying functions in instance-based learning, and proposes a method based on de-activating old instances when similar new ones are available, and selectively re-activating those that are consistent with new data. Ideally, augmenting CBR systems with the ability to detect regularity problems and respond to problem trends will improve their ability to avoid future failures and organize their case bases for efficient access.

## 3.3 Calculating the Regularity Values

In order to apply the formulas to trigger maintenance, practical means are needed to calculate their values. Because *ProbDistReg* depends only on the levels of similarity between new problems and the cases retrieved to deal with them (which are available as a byproduct of normal processing), *ProbDistReg* can be calculated easily.

On the other hand, calculating *ProbSolnReg* is problematic, because calculating *RDist* requires complete information about the "right" retrievals. If this information could be calculated inexpensively at retrieval time, the system could always make perfect retrievals. Nevertheless, it is sometimes possible to take advantage of information available after a problem is solved to estimate whether the right case was retrieved. The ROBBIE system [Fox and Leake 1995b], for example, detects problems in its similarity criteria by first solving the current problem, and then using the solution as the index for another retrieval, to determine if the solution from another case is more similar to the final result. If so, perfect similarity criteria would have favored that case, so the failure to retrieve it shows a flaw in problem-solution regularity.[2]

Alternatively, *ProbSolnReg* calculations could be done off-line at times when high processing cost is acceptable, to trigger off-line maintenance to improve future on-line performance.

[2]This approach does not apply to all domains, however. For example, if solutions are a single numeric value, the fact that a case in memory happens to have the correct value may be coincidental. If a CBR system estimates the price of a bunch of carrots based on the price of a bunch bought the week before, even if its estimate is wrong it is probably not appropriate to adjust its similarity to consider the carrots more similar to a light bulb that happens to cost precisely the correct amount.

## 3.4 Using the Formulas as Maintenance Triggers

The previous definitions provide a basis for judging the levels of regularity for particular systems, case bases, and problem sequences. By monitoring the levels of regularity and their changes, it is possible to identify needs for maintenance. For example,

- When problem-solution similarity falls below acceptable levels, it may signal:
  - Failure of the similarity metric to capture features that have become important in predicting *RDist* for current problems (e.g, if a route planner does not consider the direction of old paths when doing retrieval, and is called upon to plan paths in a new area with many one-way streets).
  - Changes in the problem-solving environment that require adjusting the solutions that would have applied to the same problems in the past, so that *RDist* itself has changed and *PDist* must be adjusted to be consistent (e.g, if roads have been closed, blocking paths that would previously have been successful).
- When problem-distribution regularity falls below acceptable levels, it may signal:
  - Insufficient case coverage of the current problems (additional cases would increase the chance of having one available within the acceptable neighborhood).
  - Flawed or insufficient adaptation knowledge (improving adaptation knowledge would increase the size of the neighborhood of cases that is usable).
- When problem-distribution regularity is high for a subset of the case base, it may signal:
  - A "hot spot" in the case base (which enables reorganizing the case base to facilitate access to active regions, or deactivating cases from less frequently used regions.)

## 3.5 Determining How to Respond: The Role of Diachronic Analysis

Once a regularity problem has been found, it is necessary to select strategies for responding. Normally, CBR systems consider only the current problem and state of

the case base when responding to processing failures (e.g., by revising the indices for a case or storing a new case with the correct solution). However, considering trends in problems may enable better response strategies. For example, knowing that problem-solution regularity has dropped from acceptable levels to a current unacceptable level is more informative than simply knowing that the level is unacceptable, because a change in performance must be caused by changes in either the problem distribution or the environment. For example, if a system for estimating building costs consistently generates estimates that are too low, that trend suggests that a general change is needed to prevent that class of failures in the future.

One response strategy is to simply update the cases in the case base (e.g., increasing the recorded prices), but this may lose useful historical information. It may also require monitoring the update history and ages of cases, in order to make sure that all cases are updated properly. Another alternative is to keep the values of cases unchanged, but to add a "lazy" maintenance rule to adjust case solutions after they have been retrieved [Leake and Wilson 1998].

Leake and Wilson [1998] describe a class of maintenance strategies that collect data over time, over a sequence of snapshots of system processing, in order to identify trends in how case-base contents and usage are changing. They call policies based on analyzing the performance of the case-base over time *diachronic maintenance policies*. Diachronic analysis is useful, for example, for determining whether coverage problems—shown by low problem-distribution regularity—should prompt the search for additional cases. If problem-distribution regularity shows an *increasing trend*, showing that the cases being processed are filling the important regions of the case base, it may suffice to simply let the normal case learning process fill the case base. However, if the level of problem-distribution regularity is low and stable, or even decreasing, steps must be taken to increase the coverage of the case library.

Diachronic analysis is also useful to find and exploit trends in problems presented to the case base. If the problems that the system must solve consistently fall within a small neighborhood, it may suggest that the system should exploit the locality of the "hot spot" by reorganizing the case base to make cases in that region easier to access. In a distributed case base, cases in the hot spot are candidates for pre-fetching. If space limitations require that some cases be deleted, for efficiency reasons the system should also focus competence-preserving deletion [Smyth and Keane 1995] on regions other than the hot spot, in order to minimize adaptation cost on likely problems by keeping the active regions more densely populated with nearby cases.

Finally, diachronic analysis is useful for monitoring and guiding the maintenance process itself: The history of maintenance operations applied will affect choices of which operations *should* be applied. For example, if maintenance has just added a large set of cases to the case base to improve problem-distribution regularity, the

choice of whether to search for still more cases should be determined by observing the effects of the new cases over some period of time, rather than simply based on the value of $ProbDistReg$ as soon as the next input problem is processed.

## 3.6 Tools for Trend Detection

Performing diachronic maintenance requires methods for detecting underlying trends in sequences of values over time. Trend detection for numeric values can be done by a number of statistical techniques. These include simple methods such as linear regression models that attempt to find the equation of the line that best fits the data as well as time series analysis techniques such as autoregressive (integrated) moving averages (ARMA/ARIMA). Research in machine learning has studied "concept drift," in which hidden changes in context over time cause learned experiences to become inaccurate (e.g., [Salganicoff 1997]). A number of techniques have been applied to concept drift problems in time ordered domains for learning hidden context [Harries *et al.* 1998; Lane and Brodley 1998], and could be applied to adjusting similarity criteria when problem-solution regularity becomes insufficient due to concept drift.

## 3.7 Two Examples: Error Trends and Hot Spots

This section illustrates the usefulness of trend-based reasoning for responding to drops in problem-solution regularity and to patterns in problem distribution.

### Addressing Solution Error Trends:

As a simple example of the use of trend detection, we show how regression techniques can augment a case-based price estimating system, in order to make its predictions more robust despite inflation. Trend-based corrections are triggered by drops in problem-solution regularity: When the solutions predicted based on similar prior problems are no longer close to the real solutions determined by feedback to the program, maintenance is performed. The method we describe is still primarily case-based, rather than regression-based: Detected trends influence case adaptation, but the primary information source is still cases.

As our case data changing over time, we selected a college summary from the

magazine *U.S. News and World Report*.[3] The data included information on 1302 colleges, with 28 features for each one (e.g., enrollment, student test scores, etc.). The task was to predict tuition costs. Because multi-year data was not available, the increase was simulated as a normal distribution of increases around an annual inflation rate.

The following simple strategy was used for detecting and responding to error trends. The system records and monitors the percent errors between retrieved cases and evaluations. A cumulative error level is maintained by summing successive error percentages, with the expectation that accumulated percent errors due to random fluctuations (both positive and negative) will remain below a reasonable threshold magnitude. If the activation level persists above the threshold value for a specified amount of time, the system triggers a statistical analysis for possible underlying error trends. In the current system, the percentage error trend is approximated by performing a simple linear regression analysis on the sequence of error data. A maintenance rule is then installed that uses the computed regression line to forecast the percentage error for the current year and modifies cases according to the predicted error value as they are retrieved.

Experiments used query samples of 5 to 20 probes from the case set for each year over a 10 to 20 year span, selecting queries by two methods. The first method constructed a random problem distribution by selecting query cases at random. The second method constructed a highly regular problem distribution by restricting the query population to a set of similar instances, according to the system's similarity metric. The samples were used as probes in their respective years, over the varying year spans. The underlying annual inflation rate was varied in separate experiments between 2 and 5 percent for each year, which fluctuated according to a random normal distribution to represent yearly variations. Average error rates were measured for the baseline (no learning), case learning alone, maintenance alone, and combined case learning/maintenance. Each experiment was repeated 10 times, each time re-selecting the query sample, to obtain results on average.

While the results did not give a precise picture of how adjustments in individual parameters affected the outcomes, a general picture did emerge. With a random problem distribution, case learning performed better than the baseline, trend-based maintenance performed better than case learning, and the combination gave equivalent or better results. With the regularized problem distribution, the combination performed best, followed by case learning, then maintenance, and finally the baseline. A representative trial with an inflation rate of 2 percent over 15 years and sample size of 5 queries/year gave the following results. The randomized distribution showed

[3] Available from http://lib.stat.cmu.edu/datasets/.

average errors of 18 percent in the baseline, 17 percent in case learning, and 14 percent in both maintenance and combined trials. The regularized distribution showed average errors of 18 percent in the baseline, 14 percent with maintenance, 13 percent with case learning, and 12 percent in the combined trials.

The experiments point to some interesting observations. First, they suggest that maintaining existing cases can be as effective as learning new cases, and that augmenting case learning with diachronic maintenance can be beneficial. Second, it is worth noting that the individual trials of maintenance alone produced highly consistent results, while the individual trials involving case learning fluctuated a great deal in producing the average. This may indicate that detecting general trends is a more stable method of dealing with change over time than case learning. Third, typical problem distributions will likely fall somewhere between the extremes of uniform sampling (where maintenance strategies alone were better than case learning) and highly focused sampling (where case learning worked better).

## Addressing Hot Spots

A second potential use of trend detection is to respond to "hot spots" in the case base. In practice, case accesses are often non-uniform. For example, a primary motivation for the development of the GizmoTapper CBR support system for Broderbund computer games was to aid the Broderbund help desk in handling the increased queries it received soon after Christmas [Watson 1997]. The problem patterns for any domain are likely to be strongly domain-specific, but if those patterns can be detected automatically the system may be able to optimize access to information that is likely to be in demand.

To observe query distribution patterns in a real-world information source, data was gathered on accesses to Indiana University web pages for various on-line information repositories. These pages provide academic information (e.g., requirements for the BA degree) as well as homework assignments, etc. A sampling of access results for a year of logs are shown in Figure 3.1, with each band reflecting the total accesses to files within the directory. (Numbers of accesses are normalized to show the percent of maximum accesses per month from January 1998 to February, 1999. Patterns that might not have been expected (but that are easily explainable) emerge. For example, department academics pages are heavily accessed in the Fall (presumably by new students), but less frequently accessed in the Spring, as students become familiar with policies, and seldom in the summer. Pages for classes offered in Spring and Fall reflect that in their accesses. Temporal patterns are not always present—no pattern is apparent in the "Types Forum" accesses at the front of the graph—but there appears to be considerable regularity.

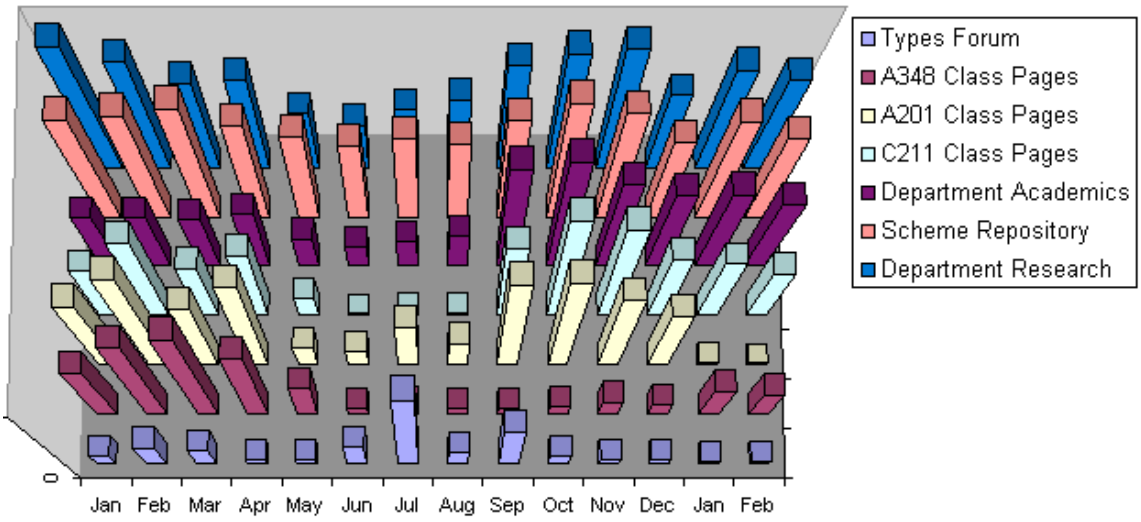



Figure 3.1: Web page accesses by month.

Various methods could be used to detect or predict hot spots, such as clustering on the problems processed, predicting problem distributions from a model of the task the CBR system serves (if available), or collecting user profiles that associate users with particular access patterns. Once a hot spot has been hypothesized, the problem-distribution regularity formula can be applied to measure the adequacy of its coverage. Insufficient coverage is a sign to examine the current problem sequence for new hot spots.

## 3.8 Considerations for Costs and Benefits

The processes described here depend on processing steps that increase the overhead of the CBR system, such as processes for trend detection and for reorganizing the case base in response to hot spots. In the long-term, more study must be done on the costs involved, but there may be important mitigating factors. First, trend analysis can be done off-line, when the system is otherwise idle. Second, in interactive CBR systems, cost and benefit analysis must weigh not only the costs incurred by the system, but also those avoided by the user. If trend analysis can, for example, warn the user of environmental changes that render prior cases obsolete, the real-world benefits may be substantial (e.g., for a realtor setting the price of a house). This may counterbalance increased computation costs.

In developing case-base maintenance strategies, it is important to have a good understanding of the maintenance problem, as well as the opportunities afforded by measures of regularity inherent in the domain and usage of a system. From the theoretical development in the last two chapters, we now move on to practical applications of maintenance techniques.

# 4

# DRAMA: Interactive Maintenance Support

Case-based reasoning is increasingly being employed in systems that incorporate significant user-interaction during problem solving (see [Aha and Muñoz-Avila 2001]). In the context of these types of systems, there are significant opportunities to develop interactive approaches to case-base maintenance.

There are many circumstances that recommend an interactive approach to case-base maintenance. For a given system, it may be that it is advantageous to incorporate explicit user interaction in the maintenance process. For example, a system that can detect a need for maintenance, but does not have the requisite domain knowledge to act upon the need, may be designed for interactive notification and support of human system maintainers. In this case, the system provides for data collection and triggering, but relies on interactive execution. Even if the system does have the ability to act upon maintenance needs, corporate strategies for knowledge management may require human management of updates to the corporate knowledge base.

When a support system is embedded in an interactive task environment, there are opportunities to learn new cases directly from users as they go about problem-solving tasks. By providing proactive support for task-oriented goals, it may be possible to help the user to construct better cases. The user views the support as helpful from a task-based perspective, and the tasks can be guided in a manner that supports good case-base maintenance practices. Thus, an important focus for case-base maintenance research is on the development of tools and methods to support human users and case-base maintainers in service of overall system goals for maintenance.

This chapter describes the DRAMA system, a case-based tool developed for aerospace design support [Leake and Wilson 2000a; 1999a]. By monitoring users as they go about normal high-level design tasks, DRAMA automatically captures

user design choices and rationale that can be used to provide proactive recommendations at both the design and design component levels. This helps to maintain the case-base through continuous support for case-authoring and design consistency while significantly ameliorating the knowledge-engineering burden on system users.

## 4.1 DRAMA Overview

Aerospace design is a complex process that requires designers to address complicated issues involving numerous specialized areas of expertise. No single designer can be an expert in every relevant area, and becoming proficient may require years of experience. Consequently, intelligent systems to support and amplify the abilities of human designers have the potential to profoundly affect the speed and reliability of design generation. An appealing approach is to augment the designers' own design experiences with relevant information from prior designs: to provide support with case-based reasoning (CBR) [Aamodt and Plaza 1994; Kolodner 1993; Leake 1996a; Riesbeck and Schank 1989; Watson 1997].

Ideally, case-based design support tools will include three related capabilities to aid reuse of designs: capture of and access to specific design experiences to enable "experience sharing" [Kitano and Shimazu 1996]; support for new designers as they try to understand the lessons of those prior experiences; and support for adapting prior designs to fit new design goals. To be practical to develop, the tools must not require the encoding of extensive domain knowledge. For designer acceptance, they must leave the designer in control. This chapter describes principles to address these goals and techniques for their application in the interactive design support framework DRAMA (Design Retrieval and Adaptation Mechanisms for Aerospace).

The DRAMA project integrates case-based reasoning with interactive tools for capturing expert design knowledge through "concept mapping" [Novak and Gowin 1984], with the goal of leveraging off the strengths of both approaches. The project uses interactive concept mapping tools, developed by the Concept Mapping group at the University of West Florida, led by Dr. Alberto Cañas, to provide an interactive interface and crucial functions for generating and examining design cases, as well as for navigating their hierarchical structure [Cañas *et al.* 1999]. Using concept maps as a case representation, as well as using the concept mapping tools to manipulate them, provide the novel capability for users themselves to develop and revise case representations. This raises interesting research questions about reconciling the conflicting goals of flexibility, customization, and case standardization as the case library grows and is maintained over time.

The implemented DRAMA system supports browsing of prior design knowledge and provides designers with concrete examples of designs and design adaptations from similar prior problems. At the same time, the DRAMA interface unobtrusively acquires new examples by monitoring the user's interactive design process. This monitoring is also used to dynamically adjust the relevance criteria for retrieving prior experiences, exploiting task-based information without requiring the user to provide it explicitly. We present experiments examining this behavior in detail, demonstrating how the benefits of this automatic retrieval approach vary under conditions modeling different levels of design novelty, different stages in the design process, and different levels of user expertise.

This chapter first presents some basic tenets that motivate the design of DRAMA's interactive framework. It then briefly summarizes case-based design and concept mapping. It goes on to describe the system itself and experimental results, followed by perspective relating its approach to other work in case-based reasoning, design, and maintenance.

## 4.2 The Task Domain

A significant concern at NASA is "knowledge loss:" that critical design expertise for their programs is the domain of a few experts and will be lost when they retire or leave the organization. This has given rise to knowledge preservation efforts, some of which have employed CBR. For example, the RECALL tool at the NASA Goddard Space Flight Center was developed to store and access textual reports of important lessons [Bagg 1997]. However, even when records have been captured they may be hard to understand and reuse. Different experts may conceptualize designs very differently, making it hard to interpret their notes of prior designs.

To make records more comprehensible, projects have investigated the use of *concept mapping* [Novak and Gowin 1984]. The goal of concept mapping for design is to capture not only important features of the designs themselves, but also designers' conceptualizations of those designs—the relationships and rationale for their components. This raises the question of how to organize and access the knowledge that concept maps capture, and how to facilitate its reuse.

This framework uses interactive CBR techniques to support retrieval and reuse of designs represented as concept maps. The processes will be illustrated with simple examples concerning high-level configuration of airliners (e.g., to select appropriate engines); a long-term goal of this work is to work with domain experts on developing richer concept maps to explore the framework as applied to design initiatives for reusable spacecraft.

## 4.3 Tenets of the Approach

The goals of the DRAMA project are twofold: First, to develop useful tools for aerospace design, and, second, to establish a general "knowledge-light" [Wilke *et al.* 1997] framework for interactive case-based design support systems. The tenets shaping this general framework are:

- *The systems developed should leverage a designer's knowledge, rather than attempting to replace it.*

  This motivates the focus on interactivity and design support rather than autonomous design. All parts of the process accept user control and interactive problem-solving.

- *Designs take many forms.*

  The system must be able to support multiple (potentially idiosyncratic) design representations, but should also encourage and support standardization when that does not impose a burden.

- *Support information should automatically be focused on the current task.*

  This requires that the system monitor the task context in order to anticipate information needs and to determine how to fulfill them.

- *Learning must play a central role, both at the design level and at the level of design manipulation.*

  This requires the capability to capture and reuse multiple types of cases.

## 4.4 Background

### Case-Based Design Support

CBR systems learn and reason by capturing and reusing lessons from analogous prior experiences. The lessons may include a wide range of information such as useful solution strategies to follow, mistaken strategies to avoid, or likely outcomes if a given strategy is followed. The fundamental process of CBR—retrieving relevant cases and using them as a guideline for new reasoning—naturally lends itself to "retrieve and propose" systems that support human reasoners by presenting the guidance of relevant prior cases [Kolodner 1991]. Many such systems have been developed and a growing number fielded [Lenz *et al.* 1998; Watson 1997].

Case-based design has been a particularly active CBR research area. It has been investigated for tasks including designing aircraft subsystems [Domeshek *et al.* 1994a], architectural design [Gebhardt *et al.* 1997; Goel *et al.* 1991; Hua and Faltings 1993; de Silva Garza and Maher 1996; Smith *et al.* 1995], circuit design [Vollrath 1998], and mechanical design [Sycara *et al.* 1991]. Design cases may record information such as the designs themselves, traces of design successes and failures, the designer's design actions, and the corresponding rationale. The case library serves as a memory of suggestions and warnings to augment the current designer's expertise. Because each suggestion is directly grounded in the experience of a prior episode, it can be explained to the current designer in terms of the results in that prior situation, and the designer can evaluate the advice by considering the relevance of the prior case to the new situation.

Many existing case-based design systems display impressive capabilities, but at the expense of considerable development effort to tailor them to domain-specific needs. The aim of DRAMA is instead to unobtrusively build up case knowledge from interactions with users as they generate designs, and to use its monitoring of the design process to extract contextual information that can be used to proactively focus retrieval on useful information as the user adapts prior cases to fit new situations. The goal is not to provide autonomous design generation or adaptation capabilities, but instead to provide useful capabilities that can be realized with limited knowledge acquisition effort.

## Concept Mapping

Concept mapping is a process designed to reveal internal cognitive structures by externalizing them in terms of networks of concepts and propositions. A concept map (CMap) is a two-dimensional representation of a set of concepts and their relationships. Individual concepts are linked to related concepts through one or two-way links, each link associated with a label/proposition describing the relationship. The vertical axis generally expresses a hierarchical framework for the concepts; for example, a concept map of design problems might represent a hierarchy of abstract and more specific problems. However, we stress that there is no requirement that they represent particular relationships; they are compatible with *any* structured representation. Semantic networks are a form of concept map, but concept maps are a more general notion: concept maps are not constrained by syntactic rules and have no associated semantics; they are normally seen as a medium for informally "sketching out" conceptual structures.

Different reasoners are likely to conceptualize a space differently, with important

variations reflected in the differences in the maps they generate. For example, a designer specializing in airflow might include features such as wing or surface shapes and operational constraints that require them (e.g., the need for short-field landings), while an avionics designer would focus on very different features. The concept mapping process is intended to help individuals to clarify their own conceptualizations and to capture them in a form that is accessible for examination by others (e.g., members of a design team seeking to understand the expert's design to evaluate or modify it, or novices seeking to increase their own understanding). One recent effort integrated concept mapping into a set of knowledge construction and sharing tools used to link over a thousand schools in Latin America [Cañas *et al.* 1995].

The visual presentation of information in concept maps provides a natural starting point for organizing and accessing information. For example, Figure 4.1 shows a sample CMap describing the basic structure of the Boeing 777 aircraft. Three nodes of this CMap are associated with images: An exterior view is accessible from the top-level node, an engine picture is associated with the engine node, and a schematic diagram of the seating layout is accessible from the nodes for any of the seating classes. The CMap is displayed by the CMap software tools to be described in the following section. Note that the maps shown in our example CMaps are high-level maps to demonstrate system capabilities; more specific maps used by expert designers would include more abstruse features or finer-grained technical details.

## Building Concept Maps

Procedures to aid the initial generation of CMaps have been described in a number of sources, primarily for use in instructional contexts to illuminate the structure of domains of study, further the synthesis of useful ideas, and encourage their analysis (e.g., [Jonassen *et al.* 1993, pp. 138–139], [Novak and Gowin 1984, pp. 24–36]). Computerized tools have also been developed to interactively support the concept mapping process. The DRAMA project uses a set of Java-based tools developed at the Institute for Human and Machine Cognition of the University of West Florida. These tools support the interactive definition and arrangement of initial maps (including multimedia components), the hierarchical examination of those maps, and transparent sharing and navigation of maps on remote servers anywhere on the Internet. The CMap tools are publicly available on the world-wide web at http://cmap.coginst.uwf.edu/.

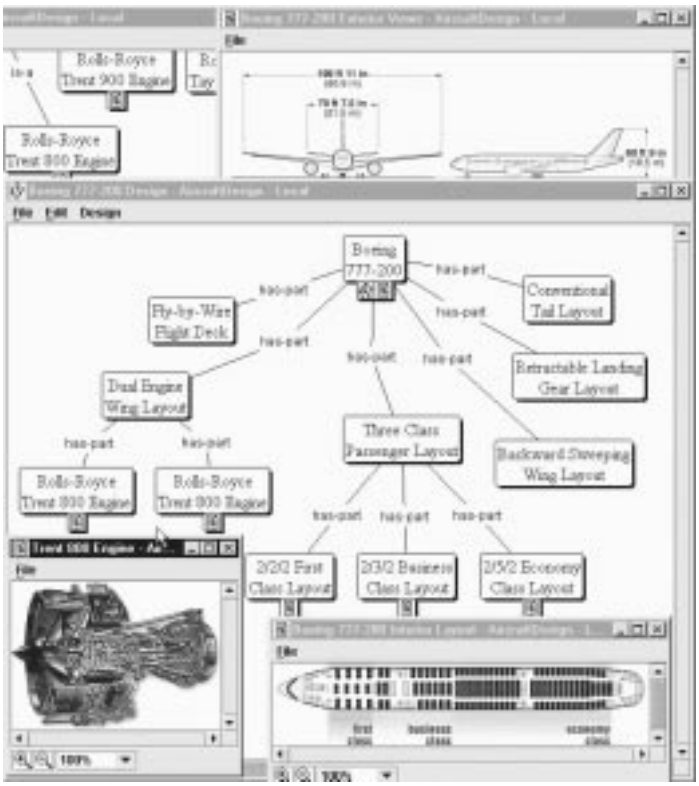



Figure 4.1: Sample screen images produced by the CMap tools.

## 4.5 Benefits of Integrating CMaps and CBR

The integration of CBR with interactive CMap tools provides leverage for both the CBR and CMap systems. Existing CMap tools provide an interactive medium for representing and browsing designs, but their framework does not provide facilities for automated searching for relevant stored CMaps. Likewise, although the tools provide capabilities for interactively defining new CMaps and manipulating their structure by adding, deleting, or substituting components, the tools provide no support for decision-making required by that adaptation process. Consequently, their usefulness can be extended by the addition of automatic facilities for retrieving relevant CMaps, automated aids to navigating CMaps and finding relevant information, and by aids to the reuse of prior CMaps.

Conversely, case-based reasoning can leverage off the interactive case definition and revision capabilities of the CMap tools. The CMap tools provide a convenient

method for entering case information in an intermediate form between textual descriptions (which are easy to input but hard to reason about) and rich structured representations (which are hard to input but support complex reasoning). In our domain, the push to use concept mapping to understand the design process is expected to make such cases available at low cost as "seed cases" for the CBR system. In addition, the tools already provide crucial functions for interactively generating and examining these cases and navigating their hierarchical structure.

## 4.6 The DRAMA System

The DRAMA system uses concept mapping tools as a method for initial capture, manual browsing, and manual modification of design cases represented as concept maps. It uses interactive CBR techniques to retrieve relevant prior cases and to retrieve alternatives to support adaptation. In addition, it uses CBR to manage and present cases recording the rationale for particular decisions and cases suggesting adaptations of designs. The following sections discuss the main features of the system.

### Using CMaps to Organize and Represent Design Information

In DRAMA, CMaps are used to represent two types of information. First, they represent hierarchies of aircraft and part types. This information is used to organize specific design cases and to guide similarity assessment during case retrieval, providing the designer with browsable hierarchies of aircraft (e.g., commercial aircraft), aircraft components (e.g., specific wings, engines, fuel tanks), and component configurations (e.g., fuel tanks inside or outside the aircraft) for reference during the design process. Our work makes no commitment to a particular set of taxonomies for a given domain. Instead it provides the tools to help particular design groups to interactively generate and refine their own sets.

Second, CMaps represent specific information about particular designs such as their components and component relationships. Each component may be represented as another CMap, enabling viewing and treating hierarchical designs at different levels of granularity through an interactive navigation process.

### How the System Supports Design

To illustrate the design process, the following sections present a simple example involving the coarse-grained configuration of an airliner after an initial set of "seed

case" designs has been provided to the system, along with hierarchies of aircraft types organizing those designs. The steps described include retrieval of a similar prior design as a starting point, retrieval support for adaptation and refinement of system suggestions, and the capture of a new adaptation for future use. Figure 4.2 summarizes these steps.

*Retrieving a relevant prior design.* The design process begins by selecting a similar example as a starting point. The user may choose either of two interfaces for the initial search process, one non-interactive and the other interactive. The first (non-interactive) option, the "Design Finder," is a simple and traditional CBR retrieval interface. The interface presents menus for selecting the desired features of a design from a pre-defined set of standard attribute types (e.g., aircraft type, manufacturer, model number). Currently the system uses a standard pre-defined feature set, but features could also be derived automatically from the set of designs.

Given the list of features, the designer selects any features of interest and the system performs nearest-neighbor retrieval, according to a predefined feature weighting scheme, to retrieve references to potentially-relevant CMaps. These are presented to the designer with a match score; the designer can browse and select from the alternatives.

The second interface allows the designer to interactively navigate the set of concept maps providing alternative "views" of aircraft and aircraft component types. This is an interactive process. As the designer proceeds through these hierarchical maps, being presented with increasingly-specific alternatives, the designer interactively determines how to proceed at each level.

In our sample scenario, the designer is considering alternatives for developing a highly fuel efficient airliner. The first step is to establish a context for the design by locating the CMap for an aircraft similar to the one envisioned. The designer then selects the engine on the CMap as the part to adapt. If no CMap is already present for the component in the starting design (e.g., the designer wishes to fill in a sketchy design by specifying its engine), the designer can use the interactive CMap tools to create a new CMap from scratch, or can browse the CMaps for designs, import a design, and then adapt as desired.

When the previous case has been retrieved, the designer has four choices, as shown in Figure 4.3: to *adapt* it (changing the representation in memory, e.g., when continuing work on a design begun in a previous session); to *derive* a new design, by having the system make a copy to adapt; to ask the system to use its hierarchy of aircraft parts to form an abstraction of the current design's structure as a template to fill in; or to ignore the design and begin a new design from scratch.

1. Retrieve prior design
   - CMap browsing
   - Manual feature-based case retrieval
2. Select working design to adapt
   - Null design (design from scratch)
   - The prior design
   - A copy of the prior design
   - A schema abstracted from the prior design
3. While working design $\neq$ target,
   (a) Select design component to adapt
   (b) Adapt component
      - Edit working component manually
      - Substitute prior component and adapt
        i. Retrieve substitute component
           - CMap browsing
           - Automatic context-based retrieval
        ii. Revise sub-parts of retrieved component recursively, starting at step 3b with the substitute component.

Figure 4.2: Steps in DRAMA's case-based design generation. Enumerated points are sequential; bullets are mutually exclusive choices within steps.

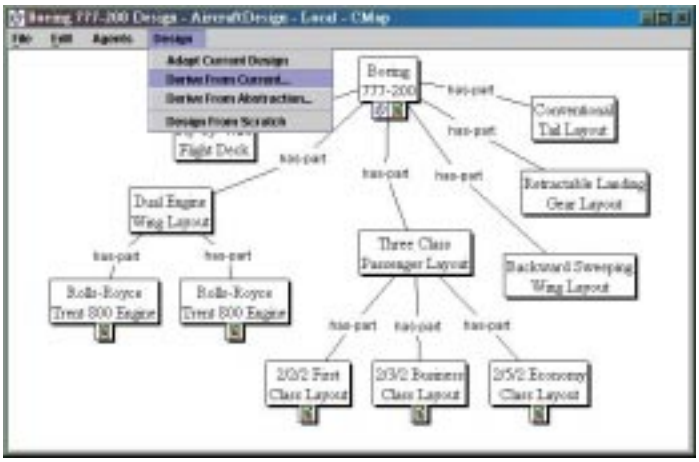



Figure 4.3: Beginning derivation of a new design from a prior case.

*Adapting designs.* The design process continues with the designer updating the current working design.

**Retrieving designs to understand or suggest new components:** Once the designer has navigated, for example, to the engine of a particular aircraft, the system supports three ways of examining why the engine was used and the alternatives that may exist. First, the designer may simply interactively browse stored information, following links in the CMap to examine associated information such as finer-grained concept maps, video clips of explanations from previous designers, or photographs or specifications of the engine.

Second, the designer may request information about similar designs. The designer may request to have this retrieval targeted to either:

- Focus on designs with components similar to the one that is currently of interest (e.g., CMaps that show aircraft using similar engines)
- Focus on designs that provide similar contexts for the current type of component (e.g., CMaps that show the engines of similar aircraft)

The algorithms underlying this retrieval are described in Section 4.7. The interface for presenting this information is shown in Figure 4.4.

Retrieved alternatives are listed in order of goodness of match according to the chosen focus. The designer may also enter additional criteria to be matched against

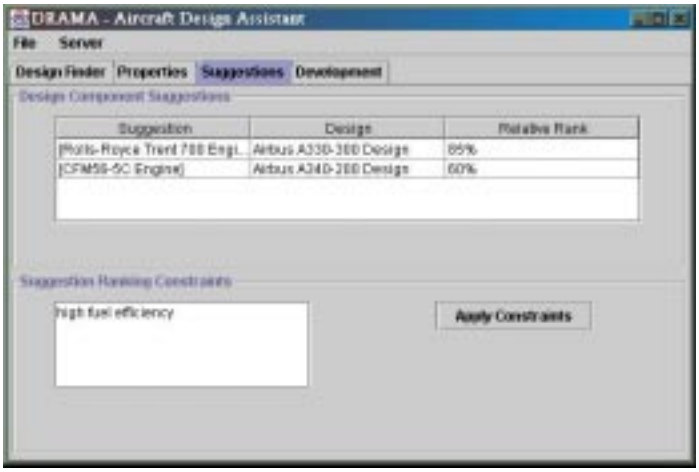



Figure 4.4: Interface for presenting engine suggestions and entering additional constraints to refine ordering by matching against textual rationale information.

any textual annotations of rationale recorded by previous designers. For example, the designer may request that fuel-efficient engines be weighted more heavily, as shown in the bottom portion of Figure 4.4. This revision uses simple text matching techniques from information retrieval (e.g., [Salton and Buckley 1988]) to decide which prior rationale to consider most relevant.

**Suggesting prior adaptations:** When the designer selects a component of an aircraft to adapt, the system has access to three pieces of information: the component affected, any designer input of additional retrieval criteria, and the design itself. This information is used to index into stored records of prior adaptations, in order to suggest proven adaptations. If adaptations have been previously performed in similar contexts to address similar issues, those adaptations are highlighted in the list of alternatives. Note that this adaptation process does not assume knowledge of complex constraints. DRAMA's method reduces the amount of knowledge needed, but at the cost of requiring the designer to evaluate the possibilities suggested (cf [Smith *et al.* 1995]).

**Performing adaptations:** When replacing an engine, the designer may select any of the suggested engines to browse further, or to substitute for the engine in the design. The designer may also simply delete or add a component to the representation using the CMap tools.

Adaptations of concept maps can be thought of as falling into three general categories corresponding to the support that they require: additions, deletions, and substitutions. Our framework supports the designer's performance of these operations as follows:

- Additions: The designer may use the plain-text retrieval capability to retrieve potentially-relevant components to be linked into the design.
- Deletions: The system can warn of potential deletion issues by proactively retrieving similar deletions, checking them for problems, and presenting those problems to the designer.
- Substitutions: The system can support substitution by retrieving and suggesting candidate substitutions, using both the explicitly-stated criteria and contextual information from the current map to guide the retrieval. It retrieves these from two sources: from stored adaptation cases encapsulating prior substitutions, and from analogous nodes in similar designs. Section 4.8 discusses experiments examining this support process.

When the designer states a goal and finds a suitable substitution, the system learns "adaptation cases" packaging the query, information about the CMap that was used as context for the search, and the selected result, following research on case-based adaptation learning [Leake *et al.* 1997b; Sycara 1987].

*Storing rationale.* After the designer performs a substitution, the designer is prompted to enter an optional textual annotation of why the new alternative is preferable to the old (e.g., an engine might have been replaced to increase fuel efficiency). Asking the designer to address this question focuses rationale capture: The designer does not record a rationale for the component as a whole (which could involve countless factors), but simply for why it is *better* than another component in the current context. Focusing the explanation process in this way is related to the common CBR idea of generating expectations for behavior and explaining only deviations from those expectations (e.g., [Hammond 1989; Ihrig and Kambhampati 1997; Schank 1982]). During future adaptations, this rationale will be provided with other information about the component, and will be used as an additional index when retrieving possible substitutions.

*Storing generated cases.* Adapted cases are placed into the system's hierarchies of cases at the point where the designer found the most similar previous case.

## 4.7 DRAMA's Method for Retrievals to Support Adaptations

Suggesting new components during adaptation requires comparing the current concept map to those in memory. Given a user-selected component (e.g., a particular engine) to be adapted, and given the goal of finding other engines from similar designs, DRAMA first retrieves a set of similar designs. The similarity assessment process begins by establishing correspondences between nodes in the old and new designs by matching their node labels. This process is equivalent to finding a matching between roles in role-filler representations. When the role appears in both old and new designs (e.g., both have a feature labeled "engine"), it is trivial to establish this correspondence. However, because concept maps are not standardized, there is no guarantee that equivalent roles will have identical labels. After pairing all the nodes which have identical labels, the matcher pairs the remaining nodes (roles) of the new design with the remaining nodes of the old design, trying to pair the roles whose fillers are most similar. For example, the "engine" role of one map might be paired with the "propulsion" role of another, because both roles are filled with engines. The matcher uses a greedy algorithm to establish this pairing while (ideally) minimizing the sum of the distances between each pair of corresponding role-fillers. In the current implementation, left-over features are ignored. Although matching cost is a potential issue, it has not proven a problem in current tests, and we have not attempted to optimize this process.

After the roles of the old and new CMaps are matched, the distance between the two CMaps is calculated by a weighted nearest-neighbor algorithm applied to the pairs of role-fillers. The distance between each pair of role-fillers is the number of links that must be traversed from one to the other in the design component abstraction hierarchy, normalized by the maximum possible distance in the hierarchy.

After retrieving concept maps for similar designs, DRAMA collects potential substitute components from each of the similar designs. The results are ranked by the inverse of each design map's distance from the current context. This gives an indication of the relative goodness of each suggestion within the overall pool of suggestions.

Once candidate concepts have been retrieved and displayed, the user can adjust the relative ranking by entering textual descriptions of desired properties—"focus features"—to compare with the properties annotating the component. The entered text is matched against any textual annotations of a suggested component; the degree of match is calculated by counting term matches in the text and normalizing this sum by the frequencies with which the term is used in all annotations. The resulting value is taken as an additional feature added into the weighted sum for design distance.

**The significance of the approach:** Ideally, case retrieval should reflect both high-level goals and concrete design features. Applied CBR systems tend to rely on the user to explicitly provide this information (whether all at once or incrementally). One method for guiding the process is to prompt the user for information, using strategically organized questions in order to minimize the number of questions that must be asked to reach a solution. This is one of the principles of successful conversational case-based reasoning, and automated methods are now proposed to improve question organization [Aha and Breslow 1997]. In some applications contexts, however, it may be quite difficult to craft these questions. Another approach to improving retrievals is for the system to learn about the importance of particular types of features to guide its future retrievals (e.g., [Fox and Leake 1995a; Zhang and Yang 1998]).

The research focus of DRAMA's retrieval is on how to automatically generate feature values from the task context. Its aim is to integrate the CBR process tightly enough into the user's task process to infer a substantial part of the needed contextual features directly from monitoring the user's task, saving the user the effort of query formulation. This approach is closely related to research on proactive case retrieval [Leake *et al.* 1999], and we believe it will become increasingly important in fielded CBR systems.

Moreover, by monitoring users as they go about normal high-level design tasks, DRAMA automatically captures user design choices and rationale that can be used to provide proactive recommendations at both the design and design component levels. This helps to maintain the case-base through continuous support for case-authoring and design consistency while significantly ameliorating the knowledge-engineering burden on system users. In section 4.9 we discuss how DRAMA supports users in maintaining vocabulary.

Because of the close coupling between the CBR system and the user's task interface, the system has access not only to the user's retrieval request (e.g., to find a substitute engine), but also to a significant part of the context surrounding the request that will determine the relevance of the retrieval (e.g., the aircraft for which the engine is needed). The designer may augment this context with information that is not available from the current step in the design process (e.g., that the goal is to find a more fuel-efficient engine that could substitute), but is not required to do so. This approach raises a number of questions about efficiency and performance, such as what levels of performance can be achieved early in the adaptation process, when the available context may differ significantly from the final design for which a component is needed. We examine such questions in the following section.

## 4.8 Testing DRAMA's Context-Based Retrieval

Experiments were performed comparing DRAMA's context-based retrieval to two baseline manual retrieval methods, for the task of finding replacement aircraft engines. This tests the benefits of DRAMA's method for performing automatic context-based retrieval (step 3(b)ii of Figure 4.2).

A set of test retrieval problems and a set of retrieval targets were generated, as described in section 4.8. The experiments compared DRAMA's retrieval to the two baseline methods according to two criteria. The first was retrieval quality, measured by the distance of the actual retrieval from a target case to retrieve, according to a nearest-neighbor distance metric (e.g., [Watson 1997]). The second was the amount of user effort they required, measured by the number of design questions (features) that the user needed to answer (specify) in order to retrieve the target.

In the first baseline method, manual *engine-based* retrieval, an engine is retrieved by directly specifying a set of engine features. A user interactively specifies desired engine features, with the best-matching engine returned for each new feature set (ties are broken arbitrarily), until a target engine (the engine in the case library that best matches the engine in the target design) is retrieved or until all feature values are specified (see Figure 4.5). In the second baseline method, manual *aircraft-based* retrieval, a user interactively specifies desired aircraft features to retrieve an aircraft similar to the target design, and the result is the engine of the retrieved aircraft.

Our experiments examine how the dependent variables of quality and efficiency are affected by four independent variables: *retrieval method* (manual engine-based, manual aircraft-based, or DRAMA's method), *user expertise* (for manual methods, this determines the user's ability to select accurate and discriminating features and appropriate values to match the target design), *point in design process* (for DRAMA's method, this determines how well the available design context approximates the target design), and the *magnitude of changes* required to transform the starting point into the new design.

### Generating Problems and Retrieval Targets

Test problems were generated by a problem generator that randomly selects a set of aircraft design cases from the case library as starting points, each to be adapted into a new design. A target aircraft design is generated by applying hand-coded perturbation rules to the features of the selected initial design. These rules (1) adjust randomly-selected features of the aircraft design, and (2) adjust related aircraft and engine features to assure consistency of the overall design. For example, a change to

an aircraft that increases passenger capacity should increase aircraft weight as well, and an increase in aircraft weight may require increasing the thrust of the engine. The rules propagate effects of design changes to produce a consistent target design. The number of independent feature changes and the magnitude of the changes are parameters of the problem generator.

The design generation process results in a new aircraft design as a whole, including desired features for the engine. In our experiments, these engine features are taken as the ideal features that the simulated designer is attempting to approximate by retrieving the closest available engine. Consequently, to select the target engine that will be used as the standard for judging retrieval quality, the problem generator uses the list of desired features to select the closest actual engine from DRAMA's case library of known engines.

## Simulating the Manual Retrieval Processes

In many interactive CBR systems, the user interacts with the case retrieval process by repeatedly selecting and answering questions from a list of candidates. These questions determine the feature values of the problem being described, which in turn are the basis for retrievals. Each time a question is answered, the system retrieves and presents the cases that best match the current case description (e.g., [Aha and Breslow 1997]). In our experiments, this interactive retrieval process is simulated by a program that plays the role of the user, repeatedly selecting questions (unspecified features whose values need to be filled in), selecting feature values to answer them, and using the feature values in a nearest-neighbor retrieval process to retrieve a candidate design from the case library. This process is summarized in Figure 4.5. In the simulated user, answers are selected based on the features of the target aircraft, because the target aircraft corresponds to the conception of the design that the simulated designer is attempting to achieve. As each question is answered, the manual retrieval method returns the closest case in memory according to a nearest-neighbor retrieval algorithm. Because we assume that the user can identify whether the retrieved engine has the desired properties, the simulated user answers additional questions until either the target engine is selected (a user success) or until values are specified for all features, in which case the last candidate case is returned as the result.

Question selection is based on a simple model of how expertise levels affect question choices and the order in which the user chooses to answer different questions. Based on a parameter for expertise level, the simulator adjusts the probability that the next question answered by the simulated user will be the maximally-discriminating unanswered question, according to a manual ordering of the expected discriminating

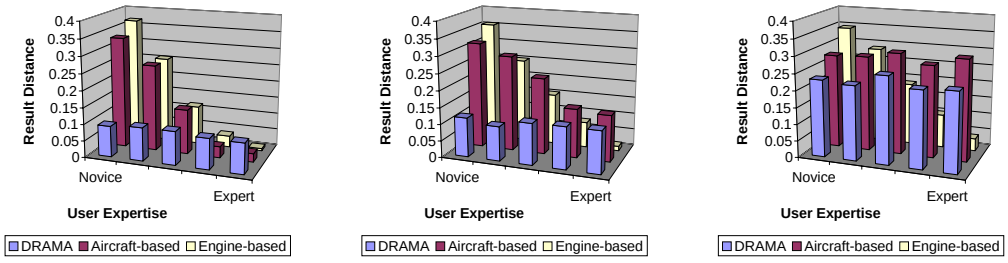
```
Initialize Q&A list
Repeat
        Select unanswered question
        Select answer based on target
        Update Q&A list
        Candidate-design =
            Retrieve-NN(Q&A list, case-base)
Until   {(Candidate-design == target) or
            (no unanswered questions)}
Return candidate-design
```

Figure 4.5: Steps in the manual retrieval process.

power for the available features. Specific questions are selected by sequentially passing over the list of unanswered questions about a particular aircraft or engine, ordered by expected discrimination power. The probability of choosing the next question in the list is given by the user's expertise level, and candidate questions are visited in order until one is chosen or the list is exhausted. If no question is chosen, an unanswered question is selected at random. Once the question is chosen, the probability that the user's answer to the question matches the target is again determined by the expertise level of the user. If the simulated user does not answer the question exactly, the answer is chosen from a distribution about the target value, based on the user's expertise level. Thus as expertise increases the possible choice distribution converges towards the target value.

**Modeling the effects of changing retrieval context:** The experiments simulate retrievals during an ongoing design process. During this process, an initial design is being adapted into a design that approximates the target. We assume that the simulated designer may adapt components of the initial design in any order; in particular, the designer may choose to adapt the initial engine—and, consequently, to retrieve the new substitute engine—at any point in the design process. This affects the quality of context that the current design provides to DRAMA's retrieval method. If the replacement engine is retrieved early, the current design will be close to the initial design, which may be quite different from the target design. On the other hand, if the replacement engine is retrieved late in the design process, the current design providing context for DRAMA's retrieval will be very close to the target design.

To test how the quality of DRAMA's retrieval depends on when retrieval occurs, our experiments generate a sequence of design steps starting with the initial design




*(a)* 1 Perturbation *(b)* 5 Perturbations *(c)* 15 Perturbations

Figure 4.6: Retrieval quality for 3 sets of perturbations of random magnitudes.

and ending with the target, reflecting the context available at each point when an engine may be retrieved. Retrievals are performed at different points in this sequence. When each retrieval is performed, the simulated user randomly selects one of the target engine's features (e.g., high fuel efficiency) as a "focus feature" to be added to DRAMA's automatically generated retrieval features. This is the only feature that the user must specify during DRAMA's retrieval, so user effort is constant.

## Experimental Setup

Tests were performed using an aircraft case library consisting of 62 cases. Each aircraft had 20 features and each engine had 12 features. Twelve of the cases were manually-entered designs representing different types of real jet aircraft (six passenger airliners ranging from small commuters to jumbo jets, two military heavy cargo planes, two military reconnaissance planes, and two military fighters) and their engines. Fifty additional designs were derived from the original twelve using the perturbation rules from the problem generator. Experiments were repeated for varying perturbation levels (numbers of changed features) and perturbation degrees (magnitude of feature-value changes). The test set consisted of 10 randomly-selected initial designs and derived targets. For each test problem, retrieval method, and level of expertise, the retrieval process was performed 10 times. Results were averaged over these 10 trials, and then averaged over the set of problems, to obtain the final reported result in each condition. Nearest-neighbor distances were normalized to values between 0 and 1.

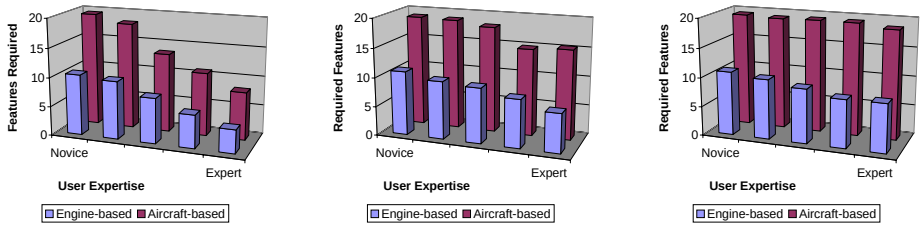
## Experimental Results

**Retrieval quality:** Predictions for comparative performance were based on expectations about the quality and types of information available to each method to

guide its retrievals. It was predicted that the retrieval quality achieved by expert users with manual engine-based retrieval methods would surpass DRAMA's method, because expert users should be able to choose a focused and accurate engine-feature context directly related to the engine type. The expectation was that DRAMA would achieve similar or somewhat better quality than expert users with aircraft-based manual methods. Experts might make better initial choices for questions to answer to retrieve relevant aircraft, but DRAMA's method has access to an additional "focus feature" to provide additional guidance about an important engine feature, and this information is not considered in aircraft-based retrieval. We also expected that DRAMA would provide better solutions than novice users: DRAMA uses all of the present available design context, which should provide a reasonably good starting point for retrieval, whereas novice users may choose features and values that comprise an inaccurate retrieval context for the desired target. Because the quality of context available to DRAMA increases as the design approaches completion, we also expected that DRAMA's retrieval quality would depend on the point in the design process at which retrievals were performed, being lowest early in the design process and increasing as the design process progressed.

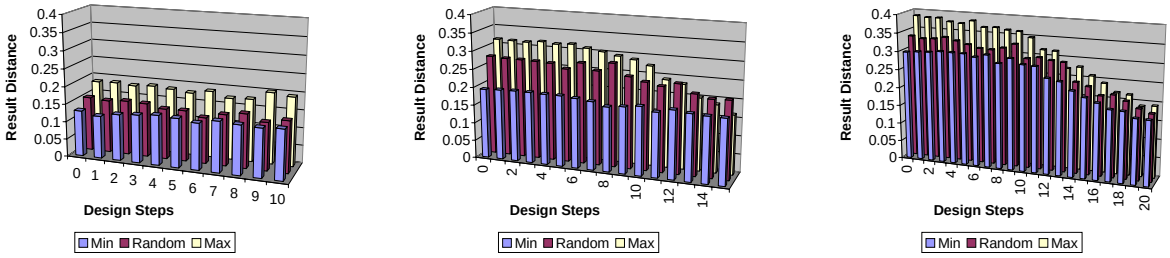
Figure 4.6 shows representative results for 3 different points in the design process for changes of random magnitude. As expected, there are crossover points determined by expertise, below which DRAMA's retrieval outperforms manual engine-based retrieval, and above which it performs worse. DRAMA's retrieval also matches or outperforms aircraft-based retrieval methods except for very low-magnitude changes in the designs. This matches expectations for most changes, for which information about the focus feature may help DRAMA discriminate between similar aircraft to select an appropriate engine. At very low-magnitude changes, the new design is sufficiently similar to the old design that we expect the marginal benefit of the extra information available to DRAMA's method—the focus feature—to be small or nonexistent. In addition, because DRAMA's retrievals are based on the current state of the design, which differs from the anticipated final design (which determines the expert's choice of features), the DRAMA method will be expected to show some limited error based on that difference, in all conditions. DRAMA's retrieval quality may be impaired by using the current design state to define retrieval features, rather than having the expert's knowledge of the target design to choose the right features to specify.

**Efficiency:** The experiments also provide information on the expected efficiency of manual retrievals for this task (measured in terms of number of questions that have to be answered to retrieve the target engine), and hence on the comparative benefits of DRAMA's method, which requires the user to specify only one focus feature. Figure 4.7 shows representative efficiency results for three perturbation levels




*(a)* 1 Perturbation *(b)* 5 Perturbations *(c)* 10 Perturbations

Figure 4.7: Efficiency for 3 sets of perturbations of random magnitudes.




*(a)* 10 Perturbations *(b)* 15 Perturbations *(c)* 20 Perturbations

Figure 4.8: Retrieval quality at successive steps in the design process for 3 sets of perturbations.

of random degree. Recall that in these tests, it is assumed that the designer can recognize the target engine as the best solution, and will continue specifying retrieval features until either that engine is selected or all features have been specified.

The relative difference between the number of questions that must be answered for aircraft- and engine- based retrieval is due to the different total numbers of aircraft and engine features (20 vs. 12). Based on our results, even experts are expected to specify a substantial number of features, although they achieve high quality retrievals with those choices. Novice users must specify many features and still achieve relatively low-quality retrievals. Because DRAMA requires only the specification of a single focus feature, DRAMA's method provides a significant reduction in the effort required to build a retrieval context while retaining reasonable quality.

It was expected that increased expertise would decrease the number of features required for the manual methods, as can be seen in Figure 4.7, graphs *(a)* and *(b)*. In some tests, however, expertise had little effect on reducing the number of features required.

**Point in the design process:** It was predicted that the greatest benefits from DRAMA's retrieval method would come from designs that were closest to completion, and that importance of closeness to completion would increase with the magnitude of the perturbations performed to generate the target design. Figure 4.8 shows the average distances of DRAMA's retrievals from the targets at design stages for minimal, random, and maximal perturbation magnitudes for three representative total numbers of perturbations. The retrieval benefits from closeness to completion are apparent in two ways. First, distances of retrievals from the target increase for targets generated using more perturbations, because the starting point of the design context is further away from the target. Second, once the context has been narrowed enough by the design approaching completion, a marked decrease in distance can be seen.

At low perturbation magnitudes, there is enough context that even moderate numbers of perturbations in generating the target do not significantly affect retrieval, so distance from the target is consistent. With higher perturbation magnitudes, after most of the perturbations in a design have been performed, the design context is sufficient for general guidance, with reasonable distances from the target, but is not specific enough to make fine distinctions. When a critical context level is achieved (here when the designs are approximately half completed), finer distinctions are made with each added piece of context, and marked improvements are seen in retrieval distance. Figure 4.8 shows that the magnitude of changes affects the importance of distance in that minimal magnitude perturbations provide a constant set of low values, while random and maximum show the reduction effect.

The three sets of experiments suggest that DRAMA's context-based retrieval can provide good suggestions with considerable savings in effort and with quality advantages over manual methods, depending on the quality of context available and on the designer's expertise in manually selecting retrieval features.

## 4.9 Perspective on Issues and Methods

The key point of the previous section is DRAMA's capability to use the context of a current design to make useful suggestions about replacement components during the design process. DRAMA's approach is also relevant to a number of fundamental issues for developing practical case-based applications. This section summarizes how the project relates to other research in those areas.

## Case Authoring

The problem of supporting case-base maintenance in the form of case authoring is receiving increasing attention from the CBR community as a whole. Ferrario and Smyth [2001] describe an approach which allows the case authoring task to be distributed across a variety of authors, and which provides support mechanisms to manage and review author submissions. It also provides a method for managing the ongoing maintenance of case-bases in dynamic domains where traditional human-based or automatic maintenance strategies prove too costly or interactive. McSherry [2001] also focuses on the case acquisition task, and presents a system that performs background reasoning on behalf of the case author while new cases are being added, in order to help the user determine the best cases to add in light of their competence contributions. The system uses its evaluations of the contributions of potential cases to suggest cases to add to the case library. DRAMA's method provides for case acquisition without requiring explicit maintenance support from the user, as such. By supporting the user in their task-oriented goals, DRAMA automatically acquires cases that are relevant to current task concerns.

## Interactive Case Acquisition

CBR is often seen as a means for overcoming the classic knowledge engineering problems associated with rule-based systems. However, successfully deploying CBR may require significant "case engineering" [Aha and Breslow 1997; Kitano and Shimazu 1996; Mark *et al.* 1996; Simoudis *et al.* 1992; Voß 1994]. Traditional structured case representations enable powerful processing at the cost of considerable case generation effort; research in textual CBR attempts to alleviate this problem by capturing information in textual form [Lenz and Ashley 1998], but must overcome problems in processing unstructured case information. Concept map representations are at a middle level between these two approaches. They include structural information, but do not enforce a standard syntax or standard set of attributes. This makes them more difficult for a CBR system to manipulate autonomously than conventional cases, but also more manageable than pure textual information. In addition, we expect this representation to facilitate non-experts in AI encoding their knowledge and to enable users to devise their own representations as needed. We have just begun to study the tradeoffs involved in this level of representation and how to address the potential problems.

For example, when cases are represented in a non-uniform way it may be difficult to identify relevant cases to retrieve. DRAMA uses two methods to help standardize CMap representations without sacrificing flexibility for the designer. First, when a

user generates a CMap and is about to fill in a new link or node, the system presents the user with a menu of alternatives from previous maps. If one of these is suitable, the user selects it. This builds up a set of standard types over time. Second, the baseline process for generating new design CMaps is modification of previous designs. The system is intended to begin with a set of CMaps that reflect the conceptualizations of a particular expert designer, reflecting that designer's coherent view of the factors important in a design. When new designs are generated by adaptation, significant portions of old representations are brought to new tasks, encouraging—but not requiring—the use of representations with similar structure. In this way, the case library and case representations are built in parallel. Practical use will give an indication of the adequacy of these methods and the overall quality of retrievals. By changing representations, in addition to maintaining the knowledge contained in the case library, this approach begins to address issues in maintaining knowledge containers other than the case-base. We discuss issues in general maintenance of knowledge containers in chapter 7.

## Guiding Design Rationale Capture

Rationale capture has long been an important topic in AI for design. A number of projects have applied rule-based or model-based approaches to the problem of design rationale capture. However, explicitly encoding all the factors relevant to a design can be prohibitively expensive, as can maintaining the required knowledge as design problems change. Consequently, it is appealing to use machine learning techniques to build up and refine design rationale knowledge incrementally.

Because CMap design cases *already* capture an entire design situation as context, we believe that useful rationale capture can be achieved with fairly limited additional information: an annotation about why the designer chose a particular component, *given the implicit context of the previous components chosen.* Although to our knowledge, this specific approach to capturing design rationale is novel, the information it captures corresponds to the "weak explanations" advocated by Gruber and Russell [Gruber and Russell 1992] in providing just enough information to guide a designer's own reasoning process. DRAMA stores this information in an unanalyzed textual form to be compared by textual matching.

Automatic capture of detailed reasoning traces has been studied in CBR research on *derivational analogy*, which captures and replays traces of the decision-making process of another knowledge-based system [Bergmann *et al.* 1998; Carbonell 1986; Veloso 1994]. Recent research has begun to apply this approach to interactive capture and replay of planning rationale to support human planners [Veloso *et al.* 1997] and for guiding search for information [Leake *et al.* 1997a]. Our project builds on these

foundations but addresses two new issues. The first is how to capture and provide useful information when it is not practical to obtain a full derivational trace. The second is how to maintain and apply case libraries from multiple designers who may have different conceptualizations of the problem and domain.

In a related spirit, but storing much more structured rationale information than DRAMA, is Clark's [Clark 1991] model of argumentation applied to geological appraisal. In that model, decisions are annotated with structured arguments (which could be represented by CMaps) aimed at helping experts to construct and improve their risk assessments. Their argumentation is a form of cooperative knowledge sharing, just as DRAMA is intended to enable designers to express, explain, and share their conceptualizations of designs.

## Conversational CBR

Conversational case-based reasoning (CCBR) systems guide the retrieval process through an interactive dialogue of questions. As described in [Aha and Breslow 1997], this approach has gained widespread acceptance in CBR shells for help desk applications; it provides both flexibility in the order of information presentation and useful incremental feedback on promising alternative cases. However, because poor questions or question organization may prevent retrieval or slow identification of the right cases, a substantial case engineering effort may be required to craft the set of questions involved, in some cases preventing the method from being applied at all.

Like CCBR, the DRAMA approach is in the spirit of interactive retrieval, but under different constraints. The prior analysis required to craft carefully targeted questions for designs would be rejected by the intended end users of DRAMA, and, at least for expert designers, the situations for which examination of prior cases is most useful are likely to be hard to anticipate. On the other hand, unlike CCBR systems used in a help desk context, for which it is necessary to build up a picture of the caller's situation, DRAMA can benefit from "free" contextual information: Because DRAMA is used as a design environment as well as a retrieval tool, it has considerable knowledge of the basic task context without the need of addressing any queries to the user.

Both DRAMA, in its initial retrieval phase, and CCBR systems are aimed at retrieving the most appropriate complete solution from previous cases. However, in its retrieval to support adaptation, DRAMA provides the ability to perform retrievals focused on subparts of the problem for the user to compose. As the user adapts part of the design, the retrieval context changes automatically, loosely corresponding to CCBR systems' adjusted rankings as more information becomes available.

## Knowledge Navigation

Case-based reasoning principles have been applied to aiding navigation and guiding interface design for web-based information sources (e.g., [Hammond *et al.* 1996; Jaczynski and Trousse 1998]), as well as for capturing and reusing knowledge about how to satisfy information needs using other information sources such as corporate databases [Fagan and Corley 1998] and hierarchical memories [Leake 1994; Leake *et al.* 1995]. All this work is relevant to the storage and reuse of information about how to navigate through a case memory of concept maps. The capability for a user to navigate directly through a hierarchy of cases, as in DRAMA's retrieval by browsing a CMap hierarchy, has recently been investigated in the HOMER system [Göker *et al.* 1998]. Interactive navigation through linked networks of cases has also been extensively studied in the context of ASK systems [Ferguson *et al.* 1992], structured hypermedia systems in which users navigate through explanatory conversations by selecting alternatives from a carefully-selected set of predefined relevant links. In contrast, the CMap framework described in this chapter focuses on retrieval when appropriate links are difficult to anticipate *a priori*.

Having described a practical interactive approach to maintaining case-base knowledge content, the next chapter moves on to describe support for maintaining representations of case-base content. Such representational maintenance is especially important for organizations in which knowledge will be accessed and employed in a wide variety of rapidly changing contexts.

# 5

# Metamorphoses: Representational Maintenance

In building CBR systems to support knowledge management and build corporate memories, it is increasingly important to be flexible in the representation of experience. Achieving widespread case-based reasoning support for corporate memories requires the flexibility to integrate various implementation-dependent representations with existing organizational resources and infrastructure. Effective techniques for maintaining case representations can be extremely useful in deploying case-based systems in many aspects of corporate experience sharing.

Constructing corporate memories that identify, acquire, and share relevant experiential knowledge across an organization is an important and challenging task. Knowledge that is stored and processed in ways optimized for one task (e.g., queries on a centralized database) may need to be used in a substantially different manner (e.g. for transfer and use over the web). Though the knowledge itself may not change, how it is collected, stored, distributed, and used may vary throughout the organization and as organizational goals change. Thus it is important to examine representational support mechanisms for varying implementation needs in building and maintaining corporate memories.

The Metamorphoses project relates three major types of representation used in CBR implementations, discusses their typical strengths and weaknesses, and describes practical strategies for building corporate CBR memories to meet new requirements by transforming and synthesizing existing resources, that is, by appropriate maintenance of the case knowledge representation.

## 5.1 Metamorphoses Overview

Achieving widespread case-based reasoning support for corporate memories will require flexibility in integrating implementations with existing organizational infrastructure and resources, as well as with new vehicles for sharing knowledge. We need to investigate not only that particular implementations address some pieces of the puzzle, but also general implementation infrastructure on which all systems can build. CBR systems as currently constructed tend to fit three general implementation models, defined by broad implementation constraints on representation and process, which reflect evolutionary developments in CBR practice. We call these models task-based, enterprise, and web-based.

Traditionally, *task-based* implementations have addressed system goals based only on the constraints imposed by the reasoning task itself. Most research systems, for example, focus on particular (often idiosyncratic) methods and representations optimized to address a specific reasoning task, either to demonstrate the effectiveness of the method or to meet specific task goals.

Recently, there has been an increasing and successful trend of incorporating CBR into enterprise systems (e.g., [Watson 1997; Stolpmann and Wess 1998]) to leverage corporate knowledge assets by knowledge management (e.g., [Becerra-Fernandez and Aha 1999]). *Enterprise* implementations reflect the additional implementation constraints imposed on CBR systems as part of an overall enterprise architecture (see [Kitano and Shimazu 1996]). In our view, the most important implementational constraint in this context is that typically such CBR integrations must operate in conjunction with database systems, the mainstay of corporate knowledge activity. This will usually mean inter-operation with the more prevalent relational database systems (e.g., [Gardingen and Watson 1998; Kitano and Shimazu 1996; Allen *et al.* 1995]), but may also include object database systems (e.g., [Ellman 1995]). Thus enterprise CBR implementations provide for and make use of database functionality. Note that not all "enterprise CBR systems" will have an enterprise implementation in this sense.

Currently, CBR systems are emerging that take advantage of recent developments in knowledge representation and sharing on the world-wide web (e.g., [Shimazu 1998; Gardingen and Watson 1998; Doyle *et al.* 1998]). *Web-based* implementations reflect additional constraints imposed on CBR systems by conforming to structured document representation standards for web/network communication, in particular XML—Extensible Markup Language [Bray *et al.* 1998]. Note that our type distinction here is based on the construction of the reasoning system itself, not on how it presents information. Thus a task-based implementation might have a web interface, and a web-based implementation might not.

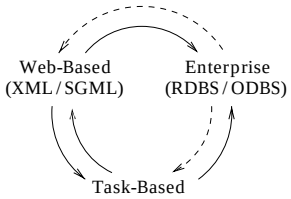



Figure 5.1: Relating CBR implementation types.

Other constraints, for example proprietary software inter-operability or Standard Generalized Markup Language (SGML) compliance, could be used to characterize enterprise or web-based implementations (and certainly more than one factor may apply). We have chosen constraints that are timely, are representative of the type of implementation concern, and that have broad applicability. As such, these implementation characterizations are intended to be useful, not perfect. They represent implementation targets in constructing corporate memories, and varying task aspects and contexts may prefer one to another. Thus it is important to understand (1) how the models compare, (2) their individual construction, (3) their combination, and especially (4) how one representation may be constructed by transforming another. Transformations are useful when new task criteria suggest a model that differs from current implementation (conversion), and when differing models are used in different aspects of a combined system (combination—e.g., database storage, web communication, task-based front end).

This chapter describes work on the Metamorphoses project to develop a framework of practical constructions and transformations, represented in Figure 5.1 (dashed lines represent additional information requirements), that we expect will play an important role in building and maintaining case-based corporate memories.

## 5.2 Implementation Models

Our implementation characterizations can be applied at many levels of typical CBR systems, and here we find it useful to differentiate CBR process knowledge containers (retrieval, adaptation, evaluation) and (case) representation knowledge. Although we recognize the importance of complex object and object-oriented database models, as well as SGML ([ISO86 1986]), here we restrict our discussion to relational database models and XML.

**Task-Based:** Task-Based implementations account for the bulk of current CBR practice. These systems allow for highly tuned and efficient metrics and representations, but it may prove difficult to reuse them outside of the system context. Some

efforts have used standardized representations to ameliorate these difficulties (e.g., [INRECA 1994]), but this is not widespread.

**Enterprise:** Integrating CBR implementations with enterprise database systems imposes standardization constraints that are almost universal in the enterprise community. Representations must accord with the table model of relational database systems (RDBS), while process must adopt Structured Query Language (SQL) conventions. CBR systems gain the strengths of the underlying RDBS, such as security, concurrency control, backup/recovery, and scalability. Moreover, integration allows the use of enterprise data both for normal corporate tasks (e.g., reporting), as well as for reasoning. However, because SQL has been designed to provide certain performance guarantees, it is limited in power, so refined metrics may be difficult to construct. Also, while complex cases are representable, they may be difficult to model in manual construction.

**Web-based:** XML is emerging as the vehicle for knowledge representation on the web. It provides a medium that allows (1) definition of customized representational markup languages and (2) application independent exchange of these complex hierarchical representations over existing web/network channels. XML also allows for customizable display of information using the associated Extensible Style Language[1]. While XML is currently viable for use (e.g., for transfer and parsing), it is a fairly new standard, so support (e.g., for browsing) is limited though growing. Its usability for applications such as CBR is also still evolving relatively rapidly [Hayes and Cunningham 1999]. Thus some benefits are immediately available for individual systems, but developing standard representations for community knowledge sharing will be a crucial task for widespread use in the field. Since XML is primarily a representation standard, it is not as tightly coupled with process as are databases, so task-based applications are generally required for process. However, direct structured query mechanisms, analogous to SQL, are under development [Sengupta 1998; W3C 1998].

## 5.3 Realizing Implementations

The realization of a framework for automatic implementation transformation involves outlining process and representation for each model, as well as defining and exemplifying the inter-model transformations. This section outlines the enterprise and web-based models (we omit the wide-ranging task-based model), and section 5.4 describes the transformations.

[1] http://www.w3.org/TR/1998/WD-xsl-19981216

## Enterprise/RDBS

Constructing an enterprise implementation involves associating a case structure with a corresponding relational database schema. Figure 5.2 shows an Entity Relationship (ER) model for typical CBR systems, where stored data represents cases (problems) which result in proposed decisions (solutions), and their outcomes (evaluations). This ER model can be fully implemented in a RDBS. The construction is straightforward for feature-vector case structures, where a single table row corresponds to a case. For more complex case structures, relational normalization techniques are used to model the data.

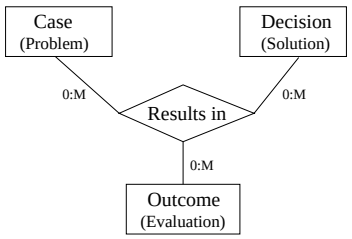



Figure 5.2: Entity Relationship diagram for a typical case-based reasoning process.

Database systems can also be used for CBR process, for example by implementing k-nearest neighbor (k-nn) retrievals. A number of novel data structures have been proposed in the database literature for efficient implementation of k-nn algorithms (e.g., [Berchtold *et al.* 1997]), but standard database systems do not currently offer such support. However, if the similarity metric can be expressed as a numeric-valued function, database cases can be retrieved as ordered by the similarity results. Thus, in our view, the database/CBR processes may take place on at least three levels:

1. *Simple Storage*: The database is used only as a storage medium. All cases are retrieved and processed by an external system. This combines the storage benefits of the database systems with task-based processing power, but requires a full task-based implementation. The basic query to the database in this case is:

   ```
   SELECT * FROM case_table
   ```

2. *Simple Retrieval*: A simple selection is performed based on conditions applied from the target, and the resulting subset is processed externally. This shifts part of the processing task to the database system, but may require considerable modeling effort to pre-compute similarity as in [Shimazu 1998], or to relax query specifications as in [Gardingen and Watson 1998; Daengdej and Lukose 1997]. The basic query here is:

```
SELECT * FROM case_table WHERE conditions
```

3. *Metric Retrieval*: A metric function is used to order the rows based on a similarity value, `metric(c)`—a function of the target case c. This uses only the database system itself to perform a full k-nn analysis. This method is inefficient, since it must both compute and sort with every record and it loses the efficiency of optimized database indexing. Thus metric retrieval has been rejected previously in principle [Shimazu *et al.* 1993], but could prove useful (given available computing power) for some (smaller-scale) implementations, since it does not require additional/external case processing for retrieval. Determining the utility of this method for a particular application requires testing in context. We have used metric retrieval with good response time in a prototype application containing 4709 cases with 24 numeric-valued features. The basic query is:

   ```
   SELECT * FROM case_table ORDER BY metric(k)
   ```

To take full advantage of database capabilities, a pre-selection of the cases in the case-base could be performed using simple retrieval before evaluating metric retrieval, to reduce (if possible by exact/ranged matching) the number of retrieved cases.

## Web-based/XML

Based on the entity-relationship model of CBR in Figure 5.2, we can also describe the structure of a full CBR system using an XML document type definition (DTD). Selected lines from the DTD are shown below:

```
<!ELEMENT CBR   (DATA, PROCESS?)>
...
<!ELEMENT DATA (PROBLEM, SOLUTION, EVAL?, RESULT?)>
<!ELEMENT PROBLEM (ATTRIBSET)>
<!ELEMENT ATTRIBSET (ATTRIB | ATTRIBSET)+>
...
<!ELEMENT PROCESS (METRIC+, ADAPT*)>
...
```

XML documents conforming to this CBR DTD describe the structure (i.e. metadata) of particular CBR systems. Components of the case base are expressed as relations (attribute sets) and their constituent attributes. Complex hierarchies are

supported by allowing sub-relations inside a relation (i.e., an ATTRIBSET inside another ATTRIBSET in the DTD). In contrast to other DTDs for CBR [Shimazu 1998; Hayes *et al.* 1998], we allow representation of both process (similarity, adaptation, evaluation) and (case) representation, either together or individually. For example, a long-term goal for this work is to develop an implementation that incorporates MathML[2], an XML DTD for describing mathematical structure and content, to represent similarity metrics.

**Using the XML model:** An instance of the above DTD describes the actual case structure, which is used by a separate XML application to generate the proper structural definition (a separate DTD) of the case data. The actual case data can then be defined as conforming instances of the generated DTD. This two-step process has the following advantages:

1. *Consistency*: By generating the case data DTD from the CBR system markup, we ensure that no separate check is necessary to assert that the structure of the case data (i.e., the separate DTD defining how the data should be structured and validated) is consistent with the data format required by the reasoning system.

2. *Validation*: Document type definitions in which the system attributes are represented as generic identifiers (tags) instead of XML attributes allow the case data to be validated against its DTD (as generated for consistency with the system) to ensure its integrity (i.e., the case data is in the form expected by the system). Given the DTD for the case data and the data itself, the validation can be performed by standard XML tools.

While XML has no particular associated process for retrieval, evolving query language implementations such as DSQL in DocBase [Sengupta 1998] and XML-QL [W3C 1998] will enhance the applicability of XML as a web-based CBR implementation model.

## 5.4 Transforming Implementations

Perhaps as important as the implementations themselves is the transformation of one implementation to another. This is useful in two situations: When new task

[2]http://www.w3.org/TR/REC-MathML/

criteria prefer a model that differs from current implementation, and when differing implementation models are used in different aspects of a combined system (e.g., database storage, web communication, task-based front end). Here we outline the transformations in the framework.

## Web-Based → Enterprise

An XML representation of case structure can be converted to a database system using an XML application that processes XML markup tags/content and generates appropriate Data Definition Language (DDL) statements to create tables in a relational database. Consider the following fragment of a CBR system description, relating a people to their automobiles:

```
<ATTRIBSET NAME="Person">
  <ATTRIB ID="ID" REQD="REQD" TYPE="longint">SSN
  </ATTRIB>
  <ATTRIB TYPE="char" SIZE="20" REQD="REQD">Name
  </ATTRIB>
  <ATTRIBSET NAME="Auto">
    <ATTRIB TYPE="char">Make</ATTRIB>
    <ATTRIB TYPE="int">Year</ATTRIB>
  </ATTRIBSET>
</ATTRIBSET>
```

By parsing this XML fragment and mapping the XML structure to relational table structure, the patterns can be translated into the following relational DDL statements:

```
create table Person (SSN longint not null,
                     Name char(20) not null);
create table Auto (Person_SSN longint not null,
                   Make char(50), Year int);
```

For complex case structures, the application can adopt a simple foreign key strategy by augmenting a substructure with the key of the parent structure. In order to facilitate a possible future back-translation, this application should also update a database catalog (organized list) with the role of each created tables in the CBR model. A similar transformation application can be used to transform XML case data to fill the generated tables.

## Web-Based → Task-Based

The main task in transforming an XML implementation to a task-based implementation is to identify a mapping between XML and task-based structures. We assume that the user or developer of the task-based systems will have the necessary tools and information to create case data in the task-based model. Taking the case representation language CASUEL [INRECA 1994] as an example, an application like the one described in section 5.4 can generate appropriate CASUEL declarations from the XML structure. This process is similar to the Web-Based→Enterprise generation process, except that the generated statements are in CASUEL instead of SQL.

## Enterprise → Web-Based/Task-Based

Transforming an existing database model into a conforming XML model or task-based model is more involved. Because the database lacks explicit case structure (when using more than a single table), transformation applications need to understand the role of various database objects in the CBR representation. Maintaining a catalog of the database objects and their roles, as suggested in section 5.4, should significantly reduce the amount of reasoning required prior to transformation. This process of role determination can be performed in several ways:

1. *Manual interaction:* The system may ask a user to assist in the process of determination of the roles of each of the objects,

2. *Catalog information:* The system may use a catalog that includes the roles of each of the objects,

3. *Mining:* The system may use data mining techniques to determine appropriate database objects and their roles.

The dashed lines in Figure 5.1 represent the extra information requirements for these transformations.

## Task-Based → Web-Based/Enterprise

Converting from task-based to an XML or database format also depends on the actual task-based model, and the availability of tools that can assist in such transformation. For example, cases represented using CASUEL can be mapped into the

corresponding XML schema or a database format using an application built on top of a CASUEL parser.

Having described techniques for maintaining case content and representation, the next chapter goes on to present practical methods for measuring a CBR system's performance in order to guide case-base maintenance processes.

# 6

# Competence & Performance

Much significant work in case-base maintenance focuses on developing methods for reducing the size of the case-base while maintaining *case-base competence*, "the range of target problems that can be successfully solved" [Smyth and McKenna 1999a]. Strategies have been developed for controlling case-base growth through methods such as competence-preserving deletion [Smyth and Keane 1995] and failure-driven deletion [Portinale *et al.* 1999], as well as for generating compact case-bases through competence-based case addition [Smyth and McKenna 1999a; Zhu and Yang 1999].

The goal of achieving compact competent case-bases addresses important performance objectives for CBR systems. First, sufficient competence is a *sine qua non* for performance: no CBR system is useful unless it can solve a sufficient proportion of the problems that it confronts. Second, compacting the case-base may help to increase system efficiency by alleviating the utility problem for retrieval [Francis and Ram 1993; Smyth and Cunningham 1996]. As an added benefit, compact case-bases decrease communications costs when case-bases are used as vehicles for knowledge sharing or are transferred in distributed CBR systems (cf. [Doyle and Cunningham 1999]).

However, case-base compactness is only a proxy for performance in a CBR system, rather than an end in itself. For example, decreased retrieval cost from a smaller case-base may be counterbalanced by increased adaptation costs or decreased quality. Thus optimizing the performance of a CBR system may require balancing tradeoffs between competence, quality, and efficiency [Portinale *et al.* 1999; Smyth and Cunningham 1996]. In addition, adjusting the case-base to optimize performance may require reasoning about the system's performance environment, taking into account that patterns in problem distribution make some cases more useful than others [Leake and Wilson 1999b]. Consequently, effective maintenance requires remembering *why* cases are being remembered (or forgotten)—to serve the overall performance goals of

the CBR system for a given task—and optimizing maintenance decisions accordingly [Leake and Wilson 1999b]. Thus performance criteria need to play a more direct role in guiding case addition and deletion.

This chapter examines the benefits of using fine-grained performance metrics to directly guide case addition and deletion, and presents some experiments on their practicality. It begins by discussing the competence/performance dichotomy and the factors that should guide case-base maintenance. It then illustrates the importance of adding direct performance considerations to maintenance strategies, by showing that in some cases, increased performance can be achieved without sacrificing either competence or compactness. It next presents a performance-based metric, guided by cases' contributions to adaptation performance, to guide case addition and deletion. Experiments examine the common alternative practice of reflecting performance with fixed adaptation effort thresholds, illuminating tradeoffs in adaptation cost and case-base compression, and then compare the effects of competence-based and performance-based strategies. Our results show that performance-based deletion strategies are especially promising for non-uniform problem distributions, which have received little attention in previous analyses of case-based maintenance, but which are often important in real-world contexts.

## 6.1 The Competence-Performance Dichotomy

Case-base maintenance is fundamentally driven by performance concerns. For example, Leake and Wilson's [1998] definition of case-base maintenance is explicitly performance-related:

> Case-base maintenance implements policies for revising the contents or organization of the case-base in order to facilitate future reasoning for a particular set of performance objectives.

In this definition, the performance measure evaluates the performance of a particular CBR system for a given initial case-base and sequence of target problems.

To relate the competence and performance of CBR systems, it is useful to revisit the notions of competence and performance. When Chomsky [1965] formulated the original competence-performance dichotomy in linguistics, he used competence to describe the "in principle" abilities of an ideal speaker, unaffected by factors such as processing limitations, and used performance to describe how language was actually used by real speakers under real constraints in real situations. "Competence" in CBR has a specialized meaning—the range of target problems that a system can

solve [Smyth and McKenna 1999a]—but the idea of "problems that a system *can* solve" can be taken to reflect an idealized competence. For example, if retrieval and adaptation time are allowed to be arbitrarily long, the competence of the case base for a sequence of input problems depends only on the "in principle" adequacy of system knowledge.

In practice, processing constraints are important, and current case-base competence research often reflects them in adaptation effort thresholds, which treat a case to be "adaptable" to solve a problem only if it can be adapted within a fixed limit on the number adaptation steps allowed (e.g., [Portinale *et al.* 1999; Zhu and Yang 1999]). Defining competence in terms of cases within the adaptation threshold combines one aspect of "idealized" competence (that the set of cases can be partitioned into adaptable and non-adaptable cases, with all adaptable cases treated as being equivalent) with the pragmatic concerns reflected in guaranteeing an upper bound on the required adaptation effort.

This chapter argues for a finer-grained approach, called *performance-based*, to make its decisions directly reflect expected impact on top-level performance goals (in these examples, goals for processing efficiency). In order to develop this approach, it first identifies the relevant performance goals and their relationships.

## 6.2 Performance Goals for Case-Base Maintenance

In general, there will be multiple performance measures for a CBR system, and there is no guarantee that all of them can be maximized simultaneously. In order to balance these measures to achieve the best overall performance, it is useful to distinguish top-level goals from goals that are only instrumental, rather than targets in themselves. For example, the goal of decreasing case-base size is not pursued for its own sake (provided space is available), but instead, as an instrumental goal of the higher-level goal to decrease retrieval time. Decreasing retrieval time is itself an instrumental goal to the top-level performance goal of improving problem-solving speed. A maintenance system that recognizes that compactness is an instrumental goal, rather than a top-level goal, can make better decisions about how to manage compactness compared to other goals, for example, by sacrificing compactness when it improves performance. However, when compactness is used as a proxy for efficiency and simply maximized, the maintenance process may miss better opportunities to maximize efficiency.

In section 2.2, it was noted that there are three types of top level goals for CBR systems:

1. Problem-solving efficiency goals (e.g., average problem-solving time)
2. Competence goals (the range of target problems solved)
3. Solution quality goals for problems solved (e.g., the error level in solutions)

It was also noted that addition and deletion strategies are also guided by the following constraints:

1. Case-base size limits (if any)
2. Acceptable long-term/short-term performance tradeoffs
3. The availability of secondary sources of cases
4. The expected distribution of future problems

Varying instantiations of these constraints would give rise to different strategies. For example, if short-term performance is crucial and long-term is less important, and current problems are concentrated in a small part of the case-base, it may be acceptable to sacrifice current competence and build it back through future learning. By their very nature, competence criteria aim at maximizing coverage, rather than trading off coverage and efficiency based on the expected problem distribution, but as we show later, making such tradeoffs may be useful for non-uniform problem distributions.

## 6.3 The Value of Performance-Based Criteria

Making the right decisions about cases to retain requires augmenting competence criteria with consideration of the performance effects of alternative cases. Usually this is thought of in terms of achieving a better tradeoff between competence and efficiency. However, in some situations, performance considerations can even improve efficiency *without loss of competence or compactness.* This is illustrated with a simple example.

For this example, it is assumed that the most easily adaptable case is always retrieved for each problem, and that the case-base is built from a set of candidates by a greedy algorithm which, for each step, adds the candidate case that provides the greatest increment to competence, until achieving full coverage [Zhu and Yang 1999]. Consider building a case-base from 3 cases, A, B, and C as shown in Figure 6.1. The line segment at the bottom of the figure represents the problem space, where

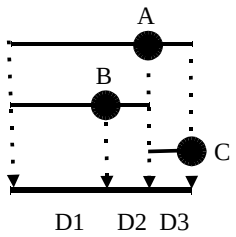



Figure 6.1: Three example cases and their coverage.

problems are associated with points on the line. For example, problems could be the desired yield strength for a metal, and solutions the manufacturing processes to obtain it. Suppose that if case $C_1$ solves problem $p_1$, the cost to adapt $C_1$ to solve a new problem $p_2$ is $\alpha|p_1 - p_2|$, for some fixed $\alpha > 0$.

The horizontal positioning of A, B, and C along the problem axis reflects the specific problems that each one solves, and the horizontal intervals adjacent to each case reflect the space of problems that it can be adapted to solve, given the system's adaptation knowledge. The interval surrounding A is an open interval on the right; case A cannot be adapted to solve the problem solved by case C. All other endpoints are closed. To build the case-base, a greedy competence-based case addition algorithm selects first case A and then case C, resulting in the case-base $CB_1 = \{A, C\}$, which provides maximal competence. We note, that $CB_2 = \{B, C\}$ provides the same competence.

If the problem distribution is uniform, it can be shown that the difference between the expected adaptation cost for solving problems using case-base $CB_1$ instead of $CB_2$ is $\alpha D_2(D_1 - D_2/4)/(D_1 + D_2 + D_3)$. If we fix $D_2$ and $D_3$ and let $D_1 \to \infty$, the expected average adaptation cost difference goes to $D_2$. Intuitively, almost all problems will then fall to the left of case B, and those problems will be $D_2$ closer to case B than to case A. Thus for this example, there are two competing case-bases with the same competence and the same size, but with different performance, so it is only possible to choose between them based on performance, not competence or compactness—and in fact, a competence-based greedy case addition algorithm picks the wrong one. This example demonstrates that performance-based considerations, distinct from competence and compactness, can play an important role in case-base selection.

## 6.4 A Performance-Based Metric for Case Selection

This section describes a strategy for performance-based case selection, inspired by Smyth and McKenna's [1999a] RC-CNN algorithm. That algorithm compacts case-bases using a compressed-nearest-neighbor (CNN) algorithm [Hart 1968] whose inputs are ordered by a relative coverage (RC) metric, to give priority to cases expected to make the largest competence contributions. By analogy to the RC metric, which estimates each case's unique contribution to the competence of the system, this work has developed a relative performance (RP) metric aimed at assessing the contribution of a case to the *adaptation performance* of the system.

### Competence & the RC Metric

The RP metric depends on two standard definitions from case-base competence research, the *coverage set* of a case (the set of problems from the target set that the problem solves) and the *reachability set* of a problem (the set of cases that solve that problem). It also depends on the *representativeness assumption* that the contents of the case-base are a good approximation of the problems the system will encounter, but can also be weighted to reflect different expected problem frequencies. The following background on basic competence definitions and the RC Metric is derived from [Smyth and McKenna 2001].

*Competence Definitions.* Competence contributions of individual cases can be characterized by two sets. The *coverage set* of a *case* is the set of all *target problems* that this case can be used to solve. It is assumed that the *Solves* predicate exists for any target CBR system. The *reachability set* of a *target problem* is the set of all *cases* that can be used to solve it. While it is not possible to enumerate all possible future target problems (T), the case-base (C) itself can be used as a representative of the target problem space to efficiently estimate these sets as shown in definitions 6.1 and 6.2.

$$CoverageSet(c \in C) = \{c' \in C : Solves(c, c')\} \tag{6.1}$$

$$ReachabilitySet(c \in C) = \{c' \in C : Solves(c', c)\} \tag{6.2}$$

The *Representativeness Assumption*—that the case-base is a representative sample of the target problem space—is fairly strong. Smyth and McKenna argue, however, that that the representativeness assumption is one currently made, albeit implicitly, by CBR researchers. If a case-base were not representative of the target problems to be solved then the system could not possibly address the task requirements.

*Relative Coverage.* The relative competence contribution of an individual case is estimated by the *relative coverage* (RC) measure, which estimates the competence contribution of an individual case, c, as a function of the size of the case's coverage set.

$$RelativeCoverage(c) = \sum_{c' \in CoverageSet(c)} \frac{1}{|ReachabilitySet(c')|} \tag{6.3}$$

RC weights the contribution of each covered case by the degree to which these cases are themselves covered. It is based on the idea that if a case c' is covered by n other cases then each of the n cases will receive a contribution of 1/n from c' to their relative coverage measures.

## The RP Metric

The RP value for a case reflects how its contribution to adaptation performance compares to that of other cases. To approximate the benefit of adding the case to the case-base, it is first assumed that the similarity metric will accurately select the most adaptable case for any problem. For each case that might be added to the case-base, its contribution to adaptation performance is estimated. A number of metrics have been explored, including a "performance benefit" (PB) metric estimating the actual numerical savings that the addition of each case provides. However, best results were obtained by considering a case's relative adaptation performance, the percent savings it provides compared to the worst alternative case that solves the problem. If we let $RS(c', c)$ stand for $ReachabilitySet(c') - \{c\}$, for a fixed case-base CB we define:

$$RP(c) = \sum_{c' \in CoverageSet(c)} 1 - \frac{AdaptCost(c, c')}{max_{c'' \in RS(c', c)} AdaptCost(c'', c')}$$

This metric can be used to guide either case addition—favoring cases with high RP values—or case deletion—favoring cases with low RP values. By adding an additional weighting factor, reflecting the expected probability of new problems similar to those in the case-base being encountered in the input stream, this formula can reflect expected problem distributions. Even if the distribution is not known completely, this adjustment can refine case selection to improve performance for likely "hot spots" in the case-base [Leake and Wilson 1999b].

Because the actual relative performance of a particular case depends on the other cases in the case-base, using completely accurate RP values to guide case deletion

would require recalculating RP values after additions or deletions, which could be extremely expensive. A more practical alternative, which we will refer to as RP-CNN, is to do a one-time RP calculation, and then to use that estimate to order the cases presented to CNN, analogously to RC-CNN. A key question is whether this approximate information is sufficiently accurate to improve performance. We test RP-CNN and compare its effects to RC-CNN in Section 6.5.

## 6.5 Experimental Results

To explore the relationships between compactification strategies and performance, four experiments were conducted. These examine (1) how the choice of adaptability thresholds affects system performance, (2) the tradeoffs between compressed case-base size and expected adaptation costs for CNN, (3) the performance obtained by RC-CNN compared to RP-CNN for uniformly-distributed problems, and (4) their comparative performance for non-uniformly distributed problems.

The experiments were conducted in a simple path planning domain that models an inter-/intra-city transportation network. Concentrated areas of local connectivity represent cities. Paths are viewed as different modes of transport between locations; they do not correspond directly to grid lines but do reflect the grid distance between location points. Models are generated randomly, based on specifications of the number and size of the cities, the number of locations in each city, the minimum distance between cities, and the maximum number of paths connecting locations. The model generator ensures that all locations are reachable through some path from all other locations, if necessary adding paths to ensure connectivity.

The planner combines case-based planning with a generative (breadth-first) path planner to adapt cases by extending their paths. This enables natural control over the allowable adaptation, by setting a threshold on the allowed number of adaptation steps. Path cases represent the starting and ending locations, the path between them, and the path distance. Cases are retrieved based on minimizing the combined distance between the starting and ending locations in a case and a new travel problem.

### Performance Effects of Competence Coverage Thresholds

Competence-preserving addition and deletion methods must determine the competence contributions of cases, which depends on the system's ability to retrieve and adapt particular cases. As described previously, the adaptability judgment is often based on an adaptation threshold, with all cases that can be adapted within the

threshold treated as equally adaptable to the problem. This blurs the adaptability differences between particular cases, sacrificing some ability to select high-performance cases. The first experiment examines the performance effects of case-bases generated by RC-CNN for different thresholds.

For each test, we generated a model consisting of 3 city areas of size 20 by 20, with 40 locations in each city. We randomly generated case-bases of sizes 1000, 750, 500, and 250 from the possible starting and ending location pairs in the model. Each case-base was then reduced in size by the RC-CNN method, and the reduced case-base was tested with 100 randomly selected probes from the model space. Each test was repeated 10 times, selecting a new initial case-set and test probes for each trial and averaging the results.

Higher threshold values increase the variance in adaptation costs for problems that a case covers, decreasing pressure to add nearby cases. Consequently, adaptation performance was expected to decrease as the threshold values increased. This basic trend appears in the results of Figure 6.2, which shows average adaptation effort on the test problems as a function of the threshold. This effect is seen across all case-base starting sizes, but lower thresholds were better at exploiting the range of cases in large case bases, selecting closer cases (resulting in lower adaptation costs). This can be explained in that all the case-bases in the experiments were large enough to provide adequate coverage, but that, at high thresholds, case choice was not sufficiently selective to take full advantage of the wider choice of cases by choosing better case distributions.

## Compressed Size vs. Adaptation Cost Tradeoffs

The previous experiment illustrates how adaptation effort thresholds used by RC-CNN can affect the adaptation effort required for a system to solve problems. However, required adaptation effort is not the only concern: There is a tradeoff because lower thresholds decrease the range of problems that each case can be used to solve, leading to the expectation that less compression is possible for a given competence level. In this experiment the effects of different case threshold levels on the case-base size obtained using RC-CNN are examined.

Using the basic experimental procedure described previously, the resulting case-base size at four different threshold levels was determined, from 1 to 10, for initial case-bases of 250, 500, 750, and 1000 cases. It was expected that as the adaptation threshold increased, the size of the case-base produced by RC-CNN would decrease. It was also expected that the resulting size would be ordered by the sizes of the case-bases, with the greatest compression being achieved for large case-bases. These

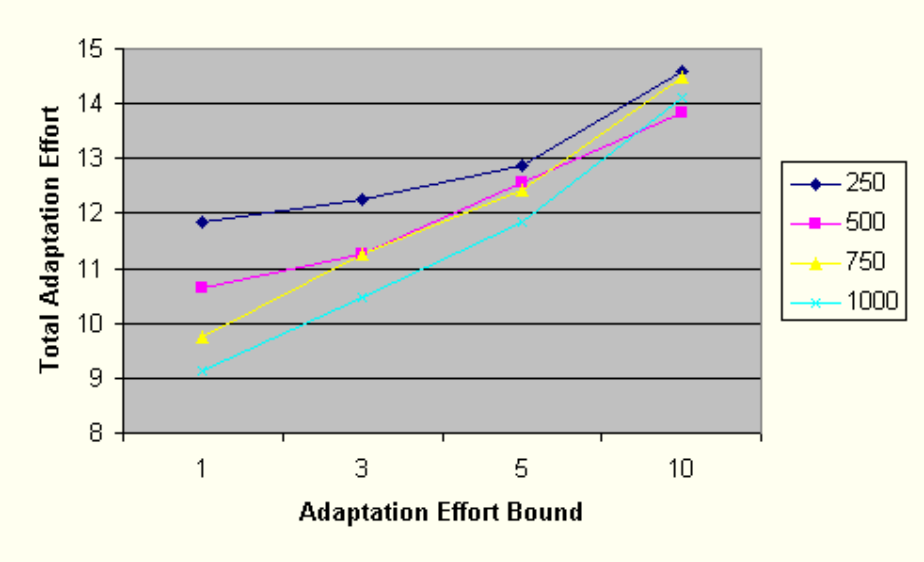



Figure 6.2: Adaptation effort as a function of threshold, for RC-CNN compression.

predictions are borne out in Figure 6.3. It is interesting to note the very substantial compression ratio achieved for a threshold of 10.

## CNN, RC-CNN and RP-CNN for Uniform Case Distributions

A third experiment compared the effects of basic CNN, RC-CNN, and RP-CNN on case-base compression and adaptation efficiency, using the same basic procedure and starting with a case-base of size 1000, with adaptation boundary of 5. For CNN, the mean case-base size was 262, for RC-CNN, 204, and for RP-CNN, 284. With a uniform distribution of test problems, mean adaptation cost for CNN was 2.96, for RC-CNN was 3.19, and for RP-CNN was 2.87. Thus as expected, RP-CNN provided some gains in efficiency at a cost of increased case-base size, while RC-CNN provided substantial gains in case-based compression at the expense of some efficiency. This provides partial independent confirmation for the results of [Smyth and McKenna 1999a].

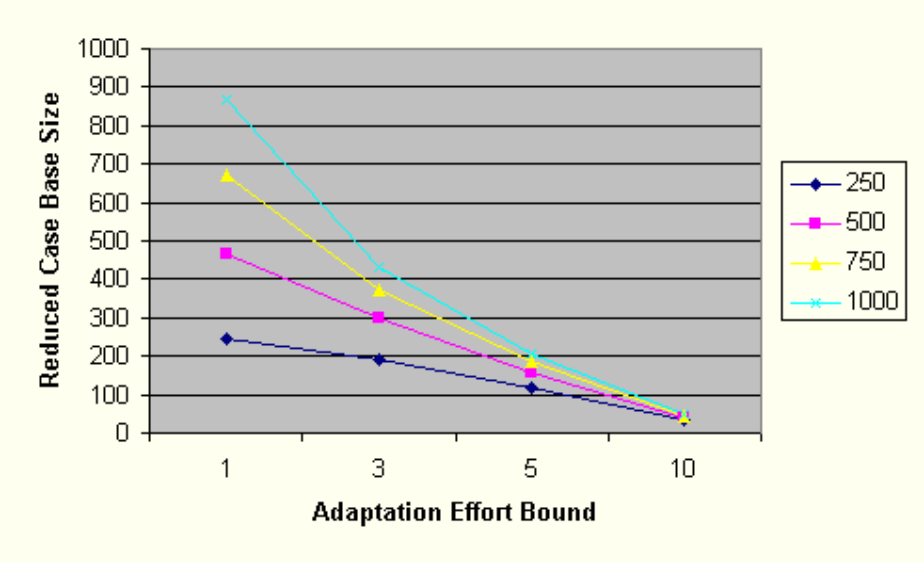



Figure 6.3: Case-base size as a function of threshold level, for RC-CNN compression.

## RC vs. RP Deletion for Non-Uniform Case Distributions

In order to test the performance of our metric under non-uniform problem distributions, an experiment was designed in which routes with origin and destination in certain cities are requested more frequently. Both the number of cities that comprise the high traffic area and the frequency of requests for routes in that area are parameters of the experiment. At the beginning of the experiment, a subset of cities of the desired size is selected at random, and routes that start from and end in those cities are considered high-traffic routes. Test probes are randomly generated from the high-traffic areas in proportion to the specified frequency, with the remaining probes randomly generated from the lower-traffic areas.

Using the same model setup as in the earlier experiments, conditions were tested in which one and two of the three cities comprised the high-traffic routes. The experiments were conducted with a 95 percent frequency rate for high-traffic probes, using the RP metric, with a weight factor to reflect the probability of a particular problem occurring (based simply on whether the problem was in a high-traffic area, and the probability of problems in that area). The effects on compression by case deletion were evaluated, first by running CNN to determine a target size for the compressed case-base, then by ordering the candidate cases according to the metric being tested (RC or RP), and deleting the least desirable cases according to the metrics,

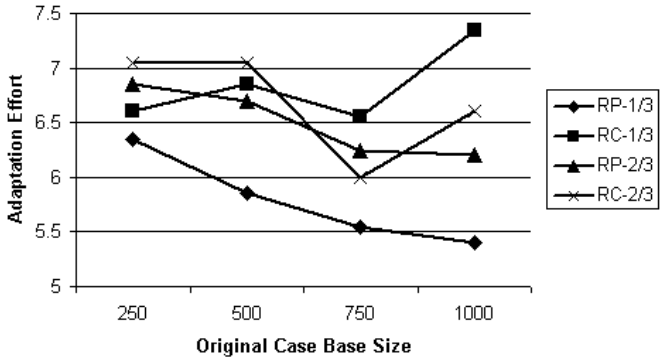



Figure 6.4: Average adaptation effort for non-uniform case distributions.

until reaching the determined size. Here it was expected to find greater performance benefits for RP than in the previous experiment, because RC focuses on coverage alone, while the revised RP favors useful cases in high-traffic areas. This was borne out in our results, which are shown in figure 6.4 for two experimental configurations: one high-traffic area and two low-traffic areas of equal size (1/3), and two high-traffic areas and one low-traffic area (2/3), for RC-CNN using an adaptation threshold of 10. The graph shows the median effort to solve cases after reduction of the case-base, for cases within the adaptation limit, for initial case-base sizes ranging from 250 to 1000 cases. For all but one test, performance with RP surpasses RC. Benefits are strongest with more focused areas (1/3), and benefits of RP appear to increase with larger initial case-bases, perhaps because the wide range of cases allows RP to fine-tune its choices.

## Performance Benefit Metric

As mentioned earlier, one of the other metrics tested was the "relative performance benefit" (PB) metric. The performance benefit value for a case is obtained by calculating the expected savings in adaptation cost from adding that case. Using the representativeness assumption, the the judgment of expected adaptation costs is based on the effects on adaptation costs when solving the problems described by the cases in the case base.

The PB metric aims to predict the performance benefit of adding a case to the case base. In formulating the definition, it is first assumed that the similarity metric will

accurately select the most adaptable case for any problem. For each case that might be added to the case base, its contribution to adaptation performance is estimated by comparing the total savings in adaptation cost (the performance benefit, $PerBenefit$) that it produces for the cases it covers, compared to the case base without that case.

The the maximum benefit for all cases in the case base is also recorded, to normalize results and obtain a value between 0 and 1 for the relative performance of that case compared to others in the case base. We let $R(c', c)$ be $ReachabilitySet(c') - \{c\}$, and define

$$PerBenefit(c) = \sum_{c' \in CoverageSet(c)} max(0, min_{c'' \in R(c',c)}(AdaptCost(c'', c') - AdaptCost(c, c')))$$

The relative performance (PB) gain from adding a case is simply the percent of the maximum possible performance benefit provided by any case in the case base:

$$PB(c) = \frac{PerBenefit(c)}{max_{c \in CB} PerBenefit(c)}$$

Note that because of the representativeness assumption, all PerBenefit values should differ from the expected average benefit by the same fixed constant, so RelativePerformance properly reflects the expected benefit of adding a case.

In experiments, the PB metric, as RP and RC, was used to order a CNN compactification of the case base. The PB-CNN tests performed worse than both RP-CNN and RC-CNN, as well as worse than straightforward CNN itself in some instances. Because the performance effectiveness of adding a new case may change significantly as new cases are added to the case-base, it is likely that the PB metric was limited by the experimental choice of the one-time calculation for CNN ordering. A long-term goal of the work is to re-examine other metrics such as PB in comparison to others using an incremental metric re-computation at each step of the compactification process.

## 6.6 Comparison to Previous Research

The importance of utility-based considerations for maintenance is well-known. Smyth and Keane's [1995] seminal competence work, for example, proposes footprint-utility deletion, in which case deletion decisions are based first on competence categories and then on utility. Smyth and Cunningham [1996] examine the tradeoffs between coverage, quality, and efficiency, illustrating how case-base size can affect retrieval and adaptation costs, as well as quality. van Someren, Surma, and Torasso

[1997] suggest using on a cost model for the CBR system to guide decisions about the size of the case-base.

Portinale, Torasso and Tavano [1998] present a case deletion strategy aimed at favoring useful cases in a combined CBR-MBR system. Their method replaces old cases with new cases solved by the MBR system, provided the new case covers the problem of the replaced case, within a fixed adaptation effort threshold, and requires more effort than the case being replaced. The *Learning by Failure with Forgetting* strategy [Portinale *et al.* 1999] applies another heuristic, periodically deleting cases that have remained unused longer than a predefined time window and "false positive cases." These are valuable heuristic methods, but differ from the RP metric's more quantitative approach, which balances the expected future performance contributions of alternative cases in the global context of competing cases in the case-base, rather than assessing cases independently.

As discussed previously, the framework here is based on the competence modeling framework of [Smyth and McKenna 1999a]. We agree with the importance of competence criteria, and advocate developing combined competence/performance metrics for tuning the maintenance process to achieve a desired balance between competence and performance concerns. For example, in a combined CBR+MBR system that can solve any problem from scratch, it may be appropriate to base maintenance decisions solely on efficiency, but in a domain where it is impossible to reconstruct deleted cases, competence concerns should receive considerable weight.

The previous chapters have developed a theoretical background for and practical approaches to case-base maintenance. The last part of the thesis takes the work that has been developed for case-base maintenance and generalizes the ideas to the general knowledge maintenance problem in case-based reasoning.

# 7

# Beyond the Case Base

It is important to note that the lessons learned from research in case-base maintenance are general, and we would like to take these lessons beyond the case-base. It has been recognized that knowledge containers other than the case-base can be equally important targets for maintenance, and the high-level goals addressed by the CBM framework are common across all the CBR knowledge containers. Multiple researchers have addressed pieces of this more general maintenance problem as well, considering such issues as how to refine similarity criteria and adaptation knowledge. As with case-base maintenance, a framework of dimensions for characterizing more general maintenance activity, within and across knowledge containers, is desirable to unify and understand the state of the art, as well as to suggest new avenues of exploration by identifying points along the dimensions that have not yet been studied.

This chapter extends and generalizes the CBM framework, presenting an overall perspective on maintaining CBR systems. It takes case-base maintenance as a starting point, adapting and generalizing the analysis of case-base maintenance to extend to other knowledge containers. The generalize framework applies across knowledge containers to characterize possible approaches to maintenance of all CBR system knowledge containers, in what we refer to as *case-based reasoner maintenance* (CBRM). CBRM processes can include not only the revision of knowledge in individual knowledge containers, but also the coordinated updating of multiple knowledge containers, the strategic transfer of knowledge between knowledge containers, and meta-maintenance—maintenance of maintenance knowledge, in order to achieve indirect revisions of knowledge containers such as lazy case-base updating.

## 7.1 A General Framework for Case-Based Reasoner Maintenance

The CBM framework developed in chapter 2 is flexible enough to provide a general set of dimensions for describing maintenance systems, beyond case-base maintenance alone. In developing the overall maintenance framework, we first define the more general *case-based reasoner maintenance*:

> Case-based reasoner maintenance implements policies for revising one or more knowledge containers in order to facilitate future reasoning for a particular set of performance objectives. [Wilson and Leake 2001]

The same CBR performance goals described in section 2.2 guide this more general process, but under more general constraints:

1. Knowledge container and component processing constraints (e.g., case-base size limits, adaptation effort thresholds)
2. Acceptable long-term/short-term performance tradeoffs
3. The availability of secondary sources of knowledge
4. The expected distribution of future problems

Regardless of the knowledge container(s) involved, maintenance policies are determined by methods for data collection, triggering, operation types, and execution. In this section we extend the framework developed for case-base maintenance to generalized case-based reasoner maintenance. As with case-base maintenance, the goals of this extended framework are to illuminate current practice by identifying classes of maintenance methods; to identify research opportunities where parts of the space of possibilities have not been addressed in previous work; and to help identify which maintenance approaches are most appropriate for particular performance goals. We do not claim that we provide a final taxonomy or a complete summary of CBRM, but that the framework provides a useful way to describe central aspects of current practice in CBRM and to identify opportunities for future maintenance research.

### Extending the CBM Framework to CBRM

Adapting the case-base maintenance framework to other knowledge containers maintains the overarching structure, but requires adjustments of some specific features. The finer-grained distinctions in operation types and triggering will change

according to the knowledge container being maintained, but both dimensions still apply. For example, operations for similarity maintenance may have different targets (e.g., weighting schemes), but the notion of revision level (implementation, representation, or knowledge level) still applies. One set of possible similarity operations is presented in [Heister and Wilke 1998].

Conditional triggering will also have different general conditions. Triggering may be based on conditions when using a particular knowledge container (e.g., the average efficiency or quality of adaptation or retrieval), or based on the results of overall reasoning. Thus the space-based CBM type of triggering may not apply for adaptation, but considerations for adaptation efficiency thresholds certainly would. We discuss issues dealing with the selection of which knowledge container to maintain in order to meet performance goals in section 7.2.

While the notion of broad vs. narrow scope still applies, the implications are relative to the particular knowledge container. In the large, revising the entire case base may take far longer than revising the entire similarity weighting scheme, but both might be considered to have broad scope within their knowledge container.

## Categorizing Policies for Other Knowledge Containers

Having extended the framework for CBRM, we present some examples of how the generalized framework may be applied to knowledge containers other than the case-base. Here we discuss policies that affect the other three well-known knowledge containers identified in [Richter 1998]: similarity, adaptation, and vocabulary. We propose, however, that the same general framework could be applied to other knowledge containers—maintenance knowledge itself, for example.

*Similarity Maintenance.* There is a large body of work dealing with methods for learning feature weighting schemes in k-nearest neighbor classifier lazy learning algorithms. Such policies typically consider a static set of training cases/instances (synchronic) in learning, are user-initiated (ad hoc), perform execution off-line (training happens off-line from system use), and are applied to the entire training set (broad). For a discussion and comparison of these types of methods, see [Wettschereck *et al.* 1997].

Muñoz-Avila and Huellen [1996] describe a policy that analyzes adaptation effort after each problem-solving episode, in order to adjust feature weights according to their relative relevance. This policy is synchronic, periodic, on-line, and makes narrow changes.

Zhang and Yang [1999] propose a method for continually updating a feature weighting scheme, based on interactive user responses to the system's behavior. This is synchronic, conditional (on receiving user feedback), on-line, and broad.

*Adaptation Maintenance.* Leake, Kinley, and Wilson [1996] present an internal case-based reasoning approach to domain-level case adaptation in the DIAL system for disaster response planning. If adaptation is required to apply a response plan, DIAL first checks for applicable adaptation cases. When no adaptation cases apply, new adaptation cases can be learned by recording traces of rule-based or interactive manual adaptation. This type of adaptation learning is synchronic, based on the system's current adaptation knowledge (i.e., if there are problems, the rule-based or manual adaptation mechanism is invoked). The timing is conditional (if the adaptation is unable to be made automatically). The activation is on-line, during case-based planning for disaster response. The scope is narrow (it applies to one adaptation case).

Hanney and Keane [1996] describe a mechanism for learning adaptation rules by induction from differences in case knowledge. If two cases differ in only a small number of attributes, then the differences in those attributes can form the basis of adaptation rules from one context to another. If there are consistent adaptation types found among potential subsets of cases, then the type of difference can be learned as adaptation rule. This policy is synchronic, ad hoc, off-line, and broad.

*Vocabulary Maintenance.* In chapter 4, we describe DRAMA, an interactive CBR system for aerospace design that uses a proactive policy to help maintain the system vocabulary (see also [Leake and Wilson 1999a]). The cases in the system are conceptual aircraft designs, for which the designers have a great deal of freedom in externalizing their design conceptualizations, freely defining new features to describe design cases. The proactive vocabulary policy examines the current design context and offers suggestions on appropriate concepts and relations that have been used previously. In this way the vocabulary is built in parallel with the case library. This policy is based on the current state of vocabulary knowledge, and is synchronic. The timing is ad-hoc, since the interactive nature of the system is under user control. The integration is on-line, and the scope is narrow.

Minor and Hanft [2000] describe an interactive framework that supports the maintenance of term dictionaries in parallel with the revision of case content. Terms are linked in with the current vocabulary (synchronic), at the user's request during processing (ad hoc and on-line), with narrow changes being made to the vocabulary.

## 7.2 Coordinating Maintenance Across Knowledge Containers

The multiple knowledge containers of CBR overlap; knowledge available in one can replace missing knowledge in another [Richter 1998]. As a result, just as builders of CBR systems can select the most convenient form in which to provide knowledge to an initial CBR system, maintainers of CBR systems can choose where to focus their maintenance efforts, applying effort where it is most convenient or effective. For example, the same overall effects on system accuracy might be achieved by case-base reorganization—which we consider part of case-base maintenance—or by adjustment of the similarity measure—which affects the similarity knowledge container. This raises two new issues for CBRM that do not arise at the individual knowledge container level: Selection of the knowledge container to maintain, and managing interactions between maintenance operations in different knowledge containers.

### Selecting the Container to Maintain

When performance goals dictate the need for maintenance, a CBRM system must determine which knowledge container(s) to revise. In some situations, only one knowledge container will be an appropriate target, while in others multiple candidate revisions could be made. For example, failure to solve a problem could be addressed by adding a new case, by adjusting similarity criteria (if the problem could have been solved starting from an existing case, but that case was not retrieved), or by adding adaptation knowledge. How to perform the credit assignment to identify which knowledge container to adjust has received some initial attention (e.g., [Leake 1996b]), for identifying indexing problems vs. missing cases), but is largely an open issue. When changes to multiple containers could be effective, utility-based choices may be needed to decide which container(s) to revise.

### Managing Interactions Between Knowledge Containers

Just as maintenance operations in one knowledge container may reduce the need for maintenance in others, maintenance in one container may necessitate maintenance in others as well. This may arise in either of two ways. First, some maintenance operations, such as revisions to the case representation vocabulary, intrinsically affect multiple knowledge containers: In order for the system to function, knowledge containers such as similarity and adaptation knowledge must be revised to handle the new representations. Thus vocabulary revisions to any container must be coupled to

associated operations to adjust the other containers. Second, maintenance operations in one knowledge container may require adjustments to other containers, in order to maintain performance or exploit revisions. For example, [Leake *et al.* 1997c] shows that realizing the benefits of augmented adaptation knowledge may depend on associated revisions of similarity criteria, in order to focus retrieval on appropriate cases for the revised adaptation knowledge. Heister and Wilke [1998] also provide a listing of the knowledge containers affected by the operations defined in their architecture.

## Case Knowledge → Adaptation Knowledge

As a first step towards examining the role of knowledge transfer in case-base maintenance, this research has investigated strategies for transferring case knowledge into adaptation knowledge. Such transfers have three possible motivations:

- Space savings, if, e.g., a small amount of added adaptation knowledge can recover the coverage of a larger number of cases, or if the adaptation knowledge can be captured in a more compact form than the cases it replaces.
- Coverage improvements, if, e.g., the new adaptation knowledge permits solution of problems beyond the ones that could be covered by the deleted case (if the new adaptation knowledge is applied to other cases in the case library).
- Problem-solving time savings, if, e.g., the decrease in case-base size provides case retrieval time savings greater than the additional cost for retrieving and applying the new adaptation knowledge.

To explore the effects of transferring case knowledge to adaptation knowledge during case-base maintenance, an experiment was performed to compare the problem-solving accuracy of a CBR system for price prediction. The Boston housing price data at the UCI Machine Learning Repository [Blake and Merz 2000] was used for testing. This data set contains 506 instances, each with 12 continuous attributes and 1 binary attribute. The adaptation learning method in the experiments was a simplified version of the strategies of [Hanney and Keane 1997] for inductive learning of adaptation rules by correlating differences in case situations with differences in their predictions. The goal of the experiments was to compare performance losses due to deletion strategies to the baseline, to test the benefits of adding knowledge transfer to case deletion strategies, and to compare the benefits of competence-preserving deletion to random deletion plus knowledge transfer. The system was tested under a number of baseline and transfer conditions. Unexpectedly, the transfer of case to adaptation knowledge showed no benefit over conditions in which case knowledge was simply compacted.

Moreover, there was a significant processing cost to adaptation rule learning. Given the relatively small size of the case-base, which was divided for training and testing, it may be that the example sets of uncovered deleted cases were insufficient for accurate learning of adaptation rules. The effectiveness of such an approach, however, was shown in concurrent research described in [Shiu *et al.* 2001]. The extra step of clustering in their approach may be a crucial step in the adaptation learning process.

This general view of case-based reasoner maintenance depends on three things: characterizing individual maintenance policies for specific knowledge containers, making strategic decisions about which strategies to apply, and characterizing how those policies are coordinated and integrated across knowledge containers, to produce effective "system-wide" maintenance procedures. In the next chapter, an example of CBRM is presented for maintaining similarity knowledge.

# 8

# CBMatrix: Similarity Maintenance

Chapter 2, illustrated diachronic case-base maintenance with an example from the CBMatrix system. In the light of a system for testing similarity maintenance techniques, this chapter, takes a closer look at case-based recommender support for scientific computing in the CBMatrix project. It begins with an overview of the CBMatrix project, discuss the role of case-based reasoning in recommender systems for scientific computing, and describe experiments for a data-structure recommendation task. In particular, it describes the use of genetic algorithms to perform similarity maintenance, by refining the similarity criteria of the CBMatrix recommender component.

## 8.1 CBMatrix Overview

Scientific problem-solving environments (PSEs) provide scientists and engineers with a framework of integrated problem-solving tools and resources that they can easily compose and apply in their particular task domains (e.g., [Gannon *et al.* 1998; Houstis *et al.* in press]). Increasingly, PSEs are being developed as applications of component architectures [Armstrong *et al.* 1999a]. Component architectures simplify and expedite the solution design process by encapsulating functional units of executable code within high-level interface specifications for composition; by facilitating the design and execution of distributed applications, enabling them to be composed from standard component services and resources; and by supporting distributed and collaborative problem-solving. Projects such as the DOE Common Component Architecture (CCA) [Armstrong *et al.* 1999a], for example, define specifications enabling scientists and engineers to write software components for high-performance computing that can be reused and composed in a wide range of computing environments. This paradigm of composing functional software units and resources shifts the focus of

problem-solving from iteratively building workable system implementations to interactively designing optimized solution strategies from existing high-level components.

Designing effective PSE solution strategies depends on making good choices about the organization and configuration of component tools and resources, and considerable expertise may be needed to achieve full benefit from the tools and resources provided by a PSE. Consequently, artificial intelligence methods to develop "recommender systems" [Kautz 1998] to guide tool selection, organization, and application have a valuable role to play in realizing the full potential of PSEs [Abelson *et al.* 1989; Gallopoulos *et al.* 1994; Ramakrishnan 1997]. In particular, the component paradigm affords significant opportunities for integrating "recommender components" that users may invoke to aid them in selecting and configuring individual components, composing multiple components, and in selecting, monitoring and managing computing resources.

CBMatrix is a research project in augmenting scientific PSEs with recommender components to support both novice and expert problem-solving. It is part of an overall effort in software component systems and PSEs for scientific computing at Indiana University (e.g., [Villacis *et al.* 1999; Gannon *et al.* 1998]), and it focuses on applications of case-based reasoning (CBR) [Kolodner 1993; Leake 1996a; Watson 1997] as the artificial intelligence methodology for making recommendations. Case-based reasoning systems reason and learn by storing records of specific prior problem-solving and re-applying their lessons in analogous situations. By unobtrusively recording the decisions of experts as they use a PSE to solve problems, and providing those decisions as suggestions to guide new problem-solving, CBR provides a vehicle for capturing and sharing expert knowledge.

The rest of this chapter describes our perspective on recommender components in scientific PSEs, presents the motivations for a case-based reasoning approach to making such recommendations, and illustrates this approach with developments in the CBMatrix project. In particular, we describe the similarity maintenance aspects of the CBMatrix recommender. Examples are drawn from our work with the Linear System Analyzer, a problem-solving environment for developing strategies to manipulate and solve large-scale sparse linear systems of equations [Gannon *et al.* 1998]. This domain is particularly good for the study of maintenance issues, as small perturbations in the environment (e.g., changes in machine usage or available memory) can have a large impact on performance.

## 8.2 Recommenders for Scientific PSEs

The goal of developing recommender components is to increase the effectiveness of problem-solving activity in PSEs. There are two important and complementary ways to further this goal, and they influence how recommender components are constructed and used. The first, *user support*, deals with helping the user to make decisions more effectively. For example, one way to support a user in setting the parameters for a linear solver might be to invoke a particular visualization tool—helping the user to understand the nature of the matrix in question, in order to select an appropriate parameterization. While the visualization tool itself is not part of the component solution strategy (linear system → parameterized solver → result), it may have played a key role in parameter choice for a similar prior problem-solving episode, and thus a user support recommender component could suggest using the visualization tool as part of current solution setup. The second, *component support*, is aimed directly at optimizing the operation of components and component compositions. For example, a component might directly recommend the best data structure representation for a sparse linear system, based on characteristics of the system, in order to achieve good performance with a given solver.

### Recommendation Types

Recommender components can be useful at all levels of scientific problem-solving, from high-level mathematical modeling to mesh manipulation to (non-)linear algebra to data analysis and visualization. Regardless of the level in question, we can typically divide possible types of recommendations into one of the following categories. Each of the following provides a suggested mapping: *user task + task context → recommendation.*

- **User Support:** Given the description of a user task and intended decision, propose resources that can provide helpful information to better enable the user's decision making. This could be applied as an alternative or addition to any of the component-based recommenders described in the following points.
- **Strategy Selection:** Given a specification of the problem to be solved, select an overall strategy for addressing the problem. For example, given a matrix, select a set of preconditioners and a solver that would give a good solution.
- **Component Selection:** Given the context (e.g., the characteristics of a differential equation or linear system to solve) in which a needed component (e.g., some linear solver) will be executed, recommend the best component for the task (e.g., a particular sparse linear solver). Note that the context could be

richer, for example, in selecting a preconditioner for a given linear system *and* a given solver.

- **Component Parameterization:** Given a component that takes parameters, and a context for that component's execution, recommend the best values for the parameter set. Note that some parameters may be fixed by the user, and would themselves become part of the context. Note also that remaining parameters could be recommended based solely on a partially specified parameter set, a kind of parameter completion recommendation.
- **Resource Selection:** Given a software component that needs to be executed, as well as the parameter settings and input to that component (or a description of the input), recommend a computational resource to use when executing the component. This could involve selecting the initial resources for a run or selecting more appropriate ones during a run, for components to automatically move themselves to more appropriate resources.

Recommenders for PSEs must also be flexible enough to support to users with varying levels of expertise, providing the information they need and shielding them from superfluous information. Recommender components should serve these goals for novices by guiding their decision-making, and for experts by providing them with advice when needed, along with explanations to help them evaluate the advice provided.

## Artificial Intelligence Recommendation Methods for Scientific Computing

An intelligent component library for scientific computing may include a range of components using different artificial intelligence methods individually or in combination. For tasks that are well understood *a priori* (e.g., selecting direct methods for dense matrices), experts can specify a set of rules that fully cover the range of recommendations associated with varying circumstances. Thus these types of recommenders leverage an existing strong domain theory. Traditional rule-based expert systems have been integrated into a number of scientific systems (e.g., for configuring PDE solver libraries [Laug 1994], for selecting elliptic PDE solution methods [Dyksen and Gritter 1992], and for selecting ODE numerical solvers [Kamel *et al.* 1992]). Because the methods rely on static pre-defined knowledge, they are considered *non-learning*.

For tasks without hard-and-fast rules (e.g., selecting preconditioned iterative methods for non-symmetric systems), techniques that learn how to make recommendations

by using sets of previous examples are more appropriate. *Eager-learning* methods (e.g., induction of decision trees, backpropagation in neural networks, and inductive logic programming) attempt to make generalizations based on a given set of examples (e.g., by deriving a set of rules). The generalizations are then used to make recommendations. This presumes that there is an implicit and relatively strong domain theory that can be exposed from the given set of examples. Research on agent-based frameworks for distributed, collaborative problem-solving and simulation [Joshi *et al.* 1997], for example, has extended and integrated earlier research on neuro-fuzzy techniques for categorization as part of addressing the algorithm selection problem for elliptic PDEs [Weerawarana *et al.* 1997]. More recently, inductive logic programming techniques have been used to learn rules for tasks such as numerical quadrature [Ramakrishnan *et al.* 2000], and analyzing performance effects in elliptic PDE solvers [Houstis *et al.* in press].

For learning tasks in which there may not be a relatively strong domain theory implicit in the working set of examples, *lazy-learning* methods (e.g., instance-based learning, case-based reasoning) that reason from specific examples instead of generalizations are more appropriate. In terms of processing, lazy-learning methods typically incur a greater on-line performance cost, but they have a much lower cost for incremental learning of new examples.

Our research concentrates on applications of case-based reasoning methods. Learning is an intrinsic part of the case-based reasoning process, because the solutions to prior problems and their outcomes are saved as cases to extend the reasoner's knowledge. When similar situations arise in the future, successful prior cases are retrieved to suggest useful reasoning to reapply, and failure cases are retrieved to warn about potential problems to avoid.

Because CBR systems can learn from single examples without requiring that those examples be generalized, CBR is an appealing method for automatically capturing information without traditional knowledge engineering. In addition, case-based reasoning can be helpful even if few examples are present. Whenever a relevant case is available, it can be applied; the system can be useful without having cases covering the entire space of potential problems. CBR has been applied to scientific computing tasks such as algorithm selection for solving elliptic PDEs [Joshi *et al.* 1996] and to guiding settings for mesh generation [Hurley 1995]. Central issues for CBR are how to index cases in memory and how to assess similarity between a new problem situation and problems solved previously, as well as how indexing and similarity criteria change and must be maintained over time.

## 8.3 CBMatrix

We have been developing a series of case-based recommender components, collectively referred to as CBMatrix. This section describes work in developing and refining one such CBMatrix component for data structure selection.

### Data Structure Recommendation

We have constructed a CBMatrix component for recommending data structures to use in solving partitioned matrix blocks from large sparse linear systems. In solving these sizable sub-matrices, efficient data structures must be used to store the individual matrix blocks while allowing standard operations to be applied effectively. The selection of an appropriate data structure for each block can significantly increase the performance of the linear solver system, speeding up the overall problem solving process. Even a small percentage improvement over standard performance could mean a significant reduction in problem solving time.

Because there are no hard-and-fast rules for selecting the best data structure, it is usually chosen based on intuitions about the sparsity pattern of the overall matrix or by simply relying on one standard representation for all problems. Applying CBR to data structure selection promises three main benefits. First, by automating the choice of appropriate data structures, it enables novices to take advantage of resulting performance gains and relieves the expert of the burden of manual data structure selection (especially when that choice would be made for each block of a partitioned matrix). Second, a CBR system can improve its performance by storing cases corresponding to expert choices and results. Third, cases can be used to inform users (to teach a novice or provide support for an expert), by explaining system recommendations with examples of similar situations.

Given a new matrix, the system recommends the data structures that were most appropriate for similar matrices solved in the past. The similarity judgment is based on easy to compute characteristic features of the matrices (e.g., number of non-zeros, degree of bandedness). The first version of our system used a weighted k-nearest neighbor algorithm (e.g., [Watson 1997]) to determine its recommendations, selecting a predetermined number of similar situations and using the results from that set to determine which data structure to suggest. Our measure of goodness was performance in flops, and the baseline data structure for performance comparisons was compressed sparse row. Tests with various methods for determining a data structure recommendation from the k-nearest neighbors indicated that the most direct method (selecting the overall closest) produced the best results. The second version of the

system used only this method. This produced good results in cross-validation testing (seeding the case-base with a portion the case data and retrieving against the rest), and mixed results for individual selections in completely new situations presented to the linear solver system. The third version implemented a similar algorithm to the second, but performed more data normalization. In cross-validation tests, the system made nearly perfect data structure selections. In informal tests using completely new situations, significant performance increases were found in approximately half of the probes.

## Refining Similarity Criteria

With encouraging results in initial tests, we were interested in how machine learning techniques might be used to refine our similarity metric. In computing similarity, it is possible for certain features to be more predictive of data structure choice (e.g., the relative number of non-zeros in a matrix is a likely candidate). If such features are known, they can be assigned a greater weight in the similarity computation. Likewise, less predictive features can be assigned a lower weight or even dropped entirely, increasing the speed of nearest-neighbor retrieval by decreasing the number of feature comparisons.

We conducted a set of tests that used genetic algorithms (GAs) [Goldberg 1989] to automatically determine a good set of feature weightings for matrix characteristics in the data structure selection task. The set of data-structure cases was divided into three distinct sets: a set to use as a reduced case-base, a set to train the GA, and a set to evaluate the weighting scheme learned by the GA. For our randomly chosen test sets, unweighted retrieval accuracy was perfect. This was excellent for the selection task itself, but obviated accuracy as a goal in evolving weight sets. The goal of minimizing the number of required features still remained, however, and the GA evolved a set of weights which preserved perfect accuracy, but reduced the number of features used at all by 63 percent (24 to 9). Taking minimization of the number of features as a new goal, the GA found a weight set that reduced the number of features by 92 percent (24 to 2) with a 7 percent loss in accuracy. In order to test whether the GA could assist in accuracy, we explicitly selected 58 cases that gave some degree of error in the unweighted condition. We performed tests that used the difficult set alone (48 training, 10 testing) and combined that set with additional randomly selected instances (152 training, 20 testing). Though the weightings in this second set of tests could not be taken to apply for the entire population, they did show showed that the GA could both improve accuracy, when gains were to be made, and reduce the number of features used.

## 8.4 Developing CBMatrix

A long-term goal of this project is to develop new case-based recommender components in conjunction with the latest component architecture research developments at Indiana University. The Indiana University Extreme! Computing Group has built CCAT [CCAT Project 2000], an implementation of the Common Component Architecture (CCA) for High Performance Computing specification [Common Component Architecture Forum 2000; Armstrong *et al.* 1999b]. CCAT has been used to implement a CCA version of the Linear System Analyzer (LSA), a problem-solving environment for developing strategies to manipulate and solve large-scale sparse linear systems of equations [Gannon *et al.* 1998; Bramley *et al.* 1998]. The LSA provides users with a palette of components that can be selected and wired together to construct complete applications. These components differ from subroutines, libraries, etc., in that component composition involves linking binaries, rather than source code to re-compile, and in that components interact on a peer-to-peer basis without one component designated as the "main" program. We expect experience gained in this framework to facilitate construction of recommender components in other PSEs.

Within the CCAT framework, components are being designed that learn by capturing parameter settings (e.g., for a particular solver), component configurations (e.g., sequences of preconditioners and solvers), and information on resource characteristics (e.g., load patterns on particular machines). Feedback on performance will be gathered both from the user (unobtrusively, for example when the user rejects suggested parameter settings) and from monitoring performance information (e.g., when reasoning from the prior case leads to expectations that conflict with observed performance). This component framework provides an interactive environment in which the context and case usefulness may easily shift over time, an ideal setting for maintenance research.

Having developed research in case-base maintenance theory and practice, and having extended case-based maintenance work to other knowledge containers, the final chapter describes conclusions and future directions for this work.

# 9

# Conclusion

This dissertation has covered theoretical and practical issues surrounding the maintenance problem in case-based reasoning, both for knowledge contained in the case-base and beyond. This chapter concludes the dissertation with a discussion of lessons learned and future directions.

## 9.1 Maintenance Framework

This work has presented a first coherent picture of the maintenance problem for knowledge contained in the case-base and extended the framework to address the general maintenance problem for knowledge containers in CBR. This general framework characterizes knowledge container maintenance policies in case-based reasoners. It presents basic dimensions for maintenance policies in terms of three subprocesses—data collection, triggering, and execution—and characterizes key design choices in terms of those dimensions. Factors considered include the type of information collected, timing, and integration of data collection; the timing and integration of maintenance triggering; whether the approach is reactive or proactive; the types of maintenance operations used; and the timing, integration, and scope of maintenance execution. The usefulness of framework has been demonstrated in describing and comparing multiple maintenance approaches, as well as in identifying new opportunities for maintenance work.

Many maintenance issues remain to be investigated. Because, to our knowledge, the role of usage trends (diachronic analysis) in guiding maintenance has not yet been explored in other research, we consider it an especially promising area. The simple trend-based maintenance described in chapters 2 and 3 has application to a particularly well-behaved type of change in the case-base that appears in other

contexts as well (e.g., updating old prices based on inflation, for real-estate appraisal) but would fail to apply to more subtle trends that would require more sophisticated methods.

Another form of trend information that might be exploited, for example, is patterns in the types of problems that are being solved. Examination of these patterns may identify "hot spots" in the problem space and determine subsets of the case-base to be consulted first, while (if storage were limited), less useful cases could be archived [Leake and Wilson 1999b]. Racine and Yang [1997] observe that recent cases may be likely to be useful; trend analysis could provide other types of suggestions for which cases should be most accessible.

Reactive maintenance has received a great deal of attention, but the reactive approach presumes that there has been some type of system failure or critical condition to which the policies are reacting. This implies that reasoning in reactive situations will not meet performance goals, or will not meet them as well. Proactive maintenance, in which policies anticipate maintenance needs before failures occur and proactively make changes, can help provide a more uniform way to continue meeting performance goals.

## 9.2 Regularity

The discussion of regularity assumptions provides steps towards understanding and responding to deviations from desired regularities. We defined measures to calculate the amount of *problem-solution regularity* and *problem-distribution regularity* that exist for the problem sequences that a case-based reasoning system encounters. The dissertation also went on to discuss methods that may be used for responding to and exploiting changing characteristics of the problems the CBR system solves, as well as of the environment in which its solutions must be applied. In particular, we described opportunities for maintenance strategies that perform their changes based on analysis of problem-solving and case-base characteristics over time—*diachronic case-base maintenance strategies.*

These definitions are useful for three reasons. First, they delineate the factors that affect regularity assumptions for CBR and their relationships—that regularity is not a property of the system or world individually but of the relationship between task, system, and the external world. Second, they provide a quantitative criterion for comparing the performance of particular CBR systems. Third, and most important, is that by giving standards for measuring regularity, they also give standards for detecting changes that require maintenance.

Determining the right response to shifting context generally requires knowledge that is unlikely to be available from a single snapshot of the CBR system's state. However, by examining *trends* in retrieval performance, system errors, and presented problems, the system may be able to respond more effectively. Sometimes, patterns of failures alone will be sufficient to hypothesize the relevant aspects of the situations and respond to changes. Even when they are not, however, a system can still respond usefully by informing the system user or maintainer of detected patterns that bear scrutiny or require investigation.

As CBR systems are more widely fielded for long-term use, it will become necessary to monitor both problem-solution regularity and problem-distribution regularity assumptions and to respond intelligently when they fail. We have provided a practical starting point for how to detect and respond to situations in which the reuse of experiences goes wrong.

## 9.3 Interactive Maintenance

The DRAMA project investigates an integrated interactive approach to case-based design support and maintenance. This approach is motivated by the complexity of aerospace design, which is such that autonomous intelligent design tools are currently infeasible, but tools that interactively support and aid the designer have promising potential.

This dissertation has described a knowledge-light framework for supporting human aerospace designers by capturing and reusing cases representing knowledge about specific prior designs, about circumstances in which those designs were useful, and about how those designs were adapted to prior needs. This framework provides natural and implicit maintenance support for case-authoring that also fosters consistency in representational vocabulary. The dissertation has also presented experimental results providing predictions of the benefit of the system's retrieval mechanism to support adaptation for different classes of users, for different levels of innovation, and at different levels of design completeness.

Knowledge-light frameworks necessarily involve tradeoffs; for example, the system can suggest components but cannot evaluate the user's designs or adaptations. Future research will investigate providing warnings when new designs match prior design cases that have known problems.

In developing DRAMA a set of principles was identified that is expected to have wide applicability to CBR integrations into interactive systems:

- Representations should be easily comprehensible and interactively adaptable by end users; visual representations may be especially useful.
- Support for representation generation should help assure consistent representations, but must not prevent the user from developing new representational elements when needed. CBR's "retrieve and adapt" process to build new cases can facilitate standardization by reusing prior representational components. This can naturally build up the case library and the representational vocabulary in parallel.
- Retrieval must tolerate representational discrepancies.
- Interactive support systems must be sufficiently integrated into the processes they support to be able to unobtrusively monitor and proactively exploit information about the task context.

The strongest overall conclusion is that interaction must be across all parts of the CBR system—initial knowledge capture, representation, retrieval, and adaptation—and across the larger task. Frameworks that allow the user and system to support each other in a shared task context, building up and using shared knowledge, have the potential to leverage off the strengths and alleviate the weaknesses of both system and user. Developing these frameworks is a challenging but promising research area.

Because the CMap tools provide the capability to share CMaps across the World Wide Web, designs from multiple designers and sites can be imported into the system's design process. Work is under way at the University of West Florida to develop CMap facilities for managing concurrent CMap generation and modification. Ideally, the design context for a particular engine, for example, could be updated as designers at other sites make changes in its specifications.

In addition to refining the system as an aid to recording and reusing design information, we see a long-term opportunity to apply it to reuse of information about *design processes*. A CMap-style interface could be used to capture traces of the steps used in generating a design (e.g., conceptual design, specification, numerical simulations, etc.), to capture how a design was formulated and to guide reasoning throughout the design process.

## 9.4 Representational Maintenance

The dissertation has presented a categorization of current CBR implementation models into three classes, and shown how this view leads to practical support for

building and maintaining case-based corporate memories by facilitating representational maintenance. The general transformations from one implementation model representation to another allow for the conversion of existing implementations and facilitate the combination of implementation types to meet new and changing task requirements.

Based on this work, we have identified three challenges for the CBR community: (1) to create community standard XML representation specifications for CBR, (2) to build a set of standard methods/libraries for translating between these XML representations and standard database representations, and (3) to develop standard CBR functionality within database systems. This will require shifting some attention to building infrastructure for the field in areas that are constrained enough to be feasible and consequential enough to be worthwhile. Community standards for representation have long been sought in many areas of AI. The structural foundations provided by web-based and enterprise media, coupled with the impetus for developing successful case-based corporate memories provides an environment ripe for achieving this goal for CBR. By providing general representational frameworks that already have ties to the world of practical application, as well as the tools to integrate them with one another and with traditional practice, the CBR community can shape the building blocks for constructing the next generation of successful research and industrial CBR systems. As CBR practice evolves, we expect the different implementation types to become increasingly integrated, and we hope to facilitate that transformation.

## 9.5 Competence & Performance

An important current of CBR research studies how to develop strategies for achieving case-bases that are competent and compact, as a proxy for good system performance. This research has presented an argument for integrating performance considerations more directly into case addition and deletion procedures, in order to allow finer-grained optimization of case-base contents. The dissertation shows that the relationship between competence, compactness and adaptation performance is more subtle than a simple tradeoff—in some circumstances, adaptation performance can be increased without sacrificing competence or compactness—motivating the search for ways to refine case addition and deletion procedures to improve performance results. It also presents empirical studies demonstrating relationships between competence criteria, adaptation performance, and case-base size, as well as an initial step towards developing a performance-guided metric for estimating the performance value of adding a case to a case-base.

Much remains to be done to refine this approach and provide a richer model in the

long-term. Such work includes refining the performance metric; performing more theoretical and empirical analyses of the tradeoffs and factors involved, considering both retrieval and adaptation costs; and combining competence and performance metrics to achieve metrics that balance both factors as desired. However, we believe that just as the direct connection of retrieval criteria to adaptation abilities led to important progress [Smyth and Keane 1998], the direct connection of case-base construction to performance criteria promises important advances for case-base maintenance research.

## 9.6 Case-Based Reasoner Maintenance

This work has take the general lessons learned from research in case-base maintenance and applied them to knowledge containers beyond the case-base. Likewise, the directions for CBRM follow those for CBM. The general view of case-based reasoner maintenance is embodied in three aspects: characterizing individual maintenance policies for specific knowledge containers, making strategic decisions about which strategies to apply, and characterizing how those policies are coordinated and integrated across knowledge containers, to produce effective system-wide maintenance policies.

Coordinating maintenance across knowledge containers is an important area for future research. Despite considerable research on maintenance policies for individual knowledge containers, there has been comparatively little investigation of how to select containers to maintain and how policies that affect more than one knowledge container interact with one another. Deciding which knowledge container to maintain and how to do so in order to address performance goals will be important in managing more complex maintenance agendas. One interesting area is knowledge container transfer, moving knowledge from one container to another to locate knowledge where it can be most easily and effectively used. Shiu et al. [2000] describe a method for transfer from the case-base to adaptation knowledge.

To exploit advances in case-based reasoner maintenance, the advances must be accompanied by increased understanding of how to apply them. A long-term goal of our work on characterizing maintenance policies is to combine the characterizations with descriptions of the tasks, domains, and performance objectives for which particular policies are likely to be appropriate, to help guide policy selection decisions when developing CBR systems.

## 9.7 Similarity Maintenance

The research has presented a perspective on recommender components for problem-solving environments in scientific computing, as well as applications and current development of case-based reasoning recommender components in the CBMatrix project. In the context of CBMatrix, it has developed techniques for maintaining the system's similarity knowledge using genetic algorithms.

The success of case-based recommender components depends on being able to select useful features for assessing the similarity of scientific computing problems. However, this burden can be ameliorated through the application of machine learning methods to enable automatic refinement of feature weightings (e.g., the GA approach described in Section 8.3). Other research issues include how to make the system adjust its recommendations in response to changes in the external processing environment (e.g., by monitoring and responding to error trends detected over time, as in [Leake and Wilson 1999b]; how to effectively access cases distributed across case libraries from different component instantiations (e.g., [Doyle and Cunningham 1999]); and how to determine which cases to retain and which to delete, in order to reduce storage requirements as large numbers of problems are solved [Leake and Wilson 1998]. A long-term goal for CBMatrix is to investigate machine learning approaches to compacting case bases through both explicit generalization of similar cases (e.g., [Domingos 1995]), and implicit generalization by choosing a smaller representative subset of cases (e.g., [Aha *et al.* 1991]). As component-based PSEs for scientific computing continue to develop, CBR will play an important role in making intelligent recommendations to support component use by experts and novices at all levels of the problem-solving process.

## 9.8 Dénouement

Further examination of the general maintenance task is a long-term research goal, both to refine our understanding and to guide the development of case-based reasoner maintenance theory and practice. The framework and practical maintenance techniques presented here build a solid foundation for further investigation, both of maintenance practice and of issues and opportunities for new maintenance approaches. The whole provides a unifying framework and algorithms for constructing and maintaining CBR systems that may be used over extended periods of time and in changing environments—a valuable resource for researchers, implementers, maintainers, and users of CBR systems; that is, a useful guide for the husbandry of experience.

# References

A. Aamodt and E. Plaza. Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AI Communications*, 7(1):39–52, 1994.

Harold Abelson, Michael Eisenberg, Matthew Halfant, Jacob Katzenelson, Elisha Sacks, Gerald Sussman, Jack Wisdom, and Kenneth Yip. Intelligence in scientific computing. *Communications of the ACM*, 32(5):546–562, 1989.

D.W. Aha and L. Breslow. Refining conversational case libraries. In *Proceedings of the Second International Conference on Case-Based Reasoning*, pages 267–278, Berlin, 1997. Springer Verlag.

David Aha and Hector Muñoz-Avila. Introduction: Interactive case-based reasoning. *Applied Intelligence*, 14(1):7–8, 2001.

D. Aha, D. Kibler, and M. Albert. Instance-based learning algorithms. *Machine Learning*, 6:37–66, 1991.

J. RC Allen, D. WR Patterson, Maurice D. Mulvenna, and John G. Hughes. Integration of case based retrieval with a relational database system in aircraft technical support. In *Proceedings of ICCBR-95*. Springer, 1995.

R. Armstrong, D. Gannon, A. Geist, K. Keahey, S. Kohn, L. McInnes, S. Parker, and B. Smolinski. Toward a common component architecture for high-performance scientific computing. In *Proceedings of the Eighth IEEE International Symposium on High Performance Distributed Computing*, 1999.

Rob Armstrong, Dennis Gannon, Al Geist, Katarzyna Keahey, Scott Kohn, Lois McInnes, Steve Parker, and Brent Smolinski. Towards a common component architecture for high-performance scientific computing. In *Proceedings of the High Performance Distributed Computing Conference*, 1999.

T. Bagg. RECALL: Reusable experience with case-based reasoning for automating lessons learned. http://hope.gsfc.nasa.gov/RECALL/homepg/recall.htm, 1997.

R. Bareiss. *Exemplar-Based Knowledge Acquisition: A Unified Approach to Concept Representation, Classification, and Learning.* Academic Press, San Diego, 1989.

I. Becerra-Fernandez and D. Aha. Case-based problem solving for knowledge management systems. In *Proceedings of the Twelfth Annual Florida Artificial Intelligence Research Symposium*, pages 219–223, Menlo Park, 1999. AAAI.

Stefan Berchtold, Christian Bohm, Daniel Keim, and Hans-Peter Kriegel. A cost model for nearest neighbor search in high-dimensional data space. In *Proceedings of the 16th ACM PODS Conference*, 1997.

R. Bergmann, H. Muñoz-Avila, M. Veloso, and E. Melis. Case-based reasoning applied to planning tasks. In M. Lenz, B. Bartsch-Spörl, H.-D. Burkhard, and S. Wess, editors, *CBR Technology: From Foundations to Applications*, chapter 7, pages 169–199. Springer, Berlin, 1998.

S. Bhatta and Ashok Goel. Model-based indexing and index learning in analogical design. In *Proceedings of the Seventeenth Annual Conference of the Cognitive Science Society*, pages 527–532, Mahwah, NJ, 1995. Lawrence Erlbaum.

C.L. Blake and C.J. Merz. UCI repository of machine learning databases, 2000.

J. Borron, D. Morales, and P. Klahr. Developing and deploying knowledge on a global scale. In *Proceedings of the Thirteenth National Conference on Artifical Intelligence*, volume 2, pages 1443–1454, Menlo Park, CA, 1996. AAAI Press.

Randall Bramley, Dennis Gannon, Thomas Stuckey, Juan Villacis, Esra Akman, Jayashree Balasubramanian, Fabian Breg, Shridhar Diwan, and Madhusudhan Govindaraju. The linear system analyzer. Technical Report TR511, Indiana University, 1998.

Tim Bray, Jean Paoli, and C. M. Sperberg-McQueen. *Extensible Markup Language 1.0*, 1998. http://www.w3.org/TR/1998/REC-xml-19980210.

A Cañas, K. Ford, J. Brennan, T. Reichherzer, and P. Hayes. Knowledge construction and sharing in quorum. In *World Conference on Artificial Intelligence in Education, AIED'95*, pages 218–225. AACE, 1995.

A. Cañas, D. Leake, and D. Wilson. Managing, mapping, and manipulating conceptual knowledge. In *Proceedings of the AAAI-99 Workshop on Exploring Synergies of Knowledge Management and Case-Based Reasoning*, pages 10–14, Menlo Park, 1999. AAAI Press.

J. Carbonell. Derivational analogy: A theory of reconstructive problem solving and expertise acquisition. In R. Michalski, J. Carbonell, and T. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, volume 2, pages 371–392. Morgan Kaufmann, Los Altos, CA, 1986.

CCAT project. <http://extreme.indiana.edu/ccat/index.html>, February 2000. Accessed February 25 2000.

W. Cheetham and J. Graf. Case-based reasoning in color matching. In *Proceedings of the Second International Conference on Case-Based Reasoning*, pages 1–12, Berlin, 1997. Springer Verlag.

N. Chomsky. *Aspects of the Theory of Syntax*. MIT Press, Cambridge, MA., 1965.

Peter Clark. *A Model of Argumentation and its Application in a Cooperative Expert System*. PhD thesis, Strathclyde University, Glascow, UK, 1991.

Common component architecture forum. <http://z.ca.sandia.gov/~cca-forum/>, February 2000. Accessed February 25 2000.

Jirapun Daengdej and Dickson Lukose. How case-based reasoning and cooperative query answering techniques support RICAD? In *Proceedings of ICCBR-97*, pages 315–324. Springer, 1997.

A. Gómez de Silva Garza and M. Maher. Design by interactive exploration using memory-based techniques. *Knowledge-Based Systems*, 9(1), 1996.

J. Deangdej, D. Lukose, E. Tsui, P. Beinat, and L. Prophet. Dynamically creating indices for two million cases: A real world problem. In I. Smith and B. Faltings, editors, *Advances in case-based reasoning*, pages 105–119, Berlin, 1996. Springer Verlag.

T. Dietterich. Learning at the knowledge level. *Machine Learning*, 1:287–316, 1986.

E. Domeshek, M. Herndon, A. Bennett, and J. Kolodner. A case-based design aid for conceptual design of aircraft subsystems. In *Proceedings of the Tenth IEEE Conference on Artificial Intelligence for Applications*, pages 63–69, Washington, 1994. IEEE Computer Society Press.

E. Domeshek, J. Kolodner, and C. Zimring. The design of a tool kit for case-based design aids. In J. Gero, editor, *Artificial Intelligence in Design*, pages 109–126, Boston, 1994. Kluwer.

P. Domingos. Rule induction and instance-based learning. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, pages 1226–1232, San Francisco, CA, August 1995. Morgan Kaufmann.

Michelle Doyle and Pádraig Cunningham. On balancing client-server load in intelligent web-based applications involving dialog. Technical Report TCD-CS-1999-25, Trinity College Dublin, 1999.

Michelle Doyle, Maria Angela Ferrario, Conor Hayes, Pádraig Cunningham, and Barry Smyth. CBR Net:- smart technology over a network. Technical Report TCD-CS-1998-07, Trinity College Dublin, 1998.

Wayne R. Dyksen and Carl R. Gritter. Scientific computing and the algorithm selection problem. In Houstis et al. [1992], pages 19–32.

Jeremy Ellman. An application of case based reasoning to object oriented database retrieval. In Ian Watson, editor, *First United Kingdom Workshop on Case-Based Reasoning*. Springer, 1995.

M. Fagan and S. Corley. CBR for the reuse of corporate SQL knowledge. In P. Cunningham, B. Smyth, and M. Keane, editors, *Proceedings of the Fourth European Workshop on Case-Based Reasoning*, pages 382–391, Berlin, 1998. Springer Verlag.

Boi Faltings. Probabilistic indexing for case-based prediction. In *Proceedings of the Second International Conference on Case-Based Reasoning*, pages 611–622, Berlin, 1997. Springer Verlag.

J. Faries and K. Schlossberg. The effect of similarity on memory for prior problems. In *Proceedings of the Sixteenth Annual Conference of the Cognitive Science Society*, pages 278–282, Atlanta, GA, August 1994. Cognitive Science Society.

W. Ferguson, R. Bareiss, R. Osgood, and L. Birnbaum. ASK systems: An approach to the realization of story-based teachers. *The Journal of the Learning Sciences*, 1:95–134, 1992.

Maria Angela Ferrario and Barry Smyth. Distributing case-base maintenance: The collaborative maintenance approach. *Computational Intelligence*, 17(2), 2001.

S. Fox and D. Leake. Modeling case-based planning for repairing reasoning failures. In *Proceedings of the 1995 AAAI Spring Symposium on Representing Mental States and Mechanisms*, pages 31–38, Menlo Park, CA, March 1995. AAAI Press.

S. Fox and D. Leake. Using introspective reasoning to refine indexing. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, pages 391–397, San Francisco, CA, August 1995. Morgan Kaufmann.

S. Fox. *Introspective Reasoning for Case-based Planning*. PhD thesis, Indiana University, 1995. Computer Science Department.

A. Francis and A. Ram. Computational models of the utility problem and their application to a utility analysis of case-based reasoning. In *Proceedings of the Workshop on Knowledge Compilation and Speed-Up Learning*, 1993.

Efstratios Gallopoulos, Elias Houstis, and John R. Rice. Computer as thinker/doer: Problem-solving environments for computational science. *IEEE Computational Science & Engineering*, 1(2):11–23, 1994.

D. Gannon, R. Bramley, T. Stuckey, J. Villacis, J. Balasubramanian, E. Akman, F. Breg, S. Diwan, and M. Govindaraju. Component architectures for distributed scientific problem solving. *IEEE Computational Science and Engineering*, 5(2):50–53, 1998.

Dan Gardingen and Ian Watson. A web based case-based reasoning system for HVAC sales support. In *Applications & Innovations in Expert Systems VI*. Springer, 1998.

F. Gebhardt, A. Voß, W. Gräther, and B. Schmidt-Belz. *Reasoning with complex cases*. Kluwer, Boston, 1997.

A. Goel, J. Kolodner, M. Pearce, and R. Billington. Towards a case-based tool for aiding conceptual design problem solving. In R. Bareiss, editor, *Proceedings*

*of the DARPA Case-Based Reasoning Workshop*, pages 109–120, San Mateo, 1991. DARPA, Morgan Kaufmann.

Mehmet Göker and Thomas Roth-Berghofer. Development and utilization of a case-based help-desk support system in a corporate environment. In K. D. Althoff, R. Bergmann, and L. K. Branting, editors, *Proceedings of the Third International Conference on Case-Based Reasoning*, pages 132–146. Springer Verlag, 1999.

M. Göker, T. Roth-Berghofer, Ralph Bergman, T. Pantleon, R. Traphöner, S. Wess, and W. Wilke. The development of HOMER: A case-based CAD/CAM help-desk support tool. In P. Cunningham, B. Smyth, and M. Keane, editors, *Proceedings of the Fourth European Workshop on Case-Based Reasoning*, pages 346–357, Berlin, 1998. Springer Verlag.

D. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*, pages 1–24. Addison-Wesley, 1989.

T. Gruber and D. Russell. Generative design rationale: Beyond the record and replay paradigm. Knowledge Systems Laboratory KSL 92-59, Computer Science Department, Stanford University, 1992.

K. Hammond, R. Burke, and K. Schmitt. A case-based approach to knowledge navigation. In D. Leake, editor, *Case-Based Reasoning: Experiences, Lessons, and Future Directions*, pages 125–136. AAAI Press, Menlo Park, CA, 1996.

K.J. Hammond. *Case-based Planning: An Integrated Theory of Planning, Learning and Memory*. PhD thesis, Yale University, 1986. Computer Science Department Technical Report 488.

K. Hammond. *Case-Based Planning: Viewing Planning as a Memory Task*. Academic Press, San Diego, 1989.

K. Hanney and M. Keane. Learning adaptation rules from a case-base. In *Proceedings of the Third European Workshop on Case-Based Reasoning*, pages 179–192, Berlin, 1996. Springer Verlag.

K. Hanney and M. Keane. The adaptation knowledge bottleneck: How to ease it by learning from cases. In *Proceedings of the Second International Conference on Case-Based Reasoning*, Berlin, 1997. Springer Verlag.

Kathleen Hanney, Mark Keane, Barry Smyth, and Padraig Cunningham. What kind of adaptation do CBR systems need? a review of current practice. In *Proceedings of the Fall Symposium on Adaptation of Knowledge for Reuse*. AAAI, 1995.

M. Harries, K. Horn, and C. Sammut. Learning in time ordered domains with hidden changes in context. In *Papers from the AAAI 1998 Workshop on Predicting the Future: AI Approaches to Time-Series Problems*, pages 29–33. AAAI, 1998.

P. E. Hart. The condensed nearest neighbor rule. *IEEE Transactions on Information Theory*, 14:515–516, 1968.

Conor Hayes and Pádraig Cunningham. Shaping a CBR view with XML. In *Proceedings of ICCBR-99*, 1999.

Conor Hayes, Pádraig Cunningham, and Michelle Doyle. Distributed CBR using XML. In *Proceedings of the KI-98 Workshop on Intelligent Systems and Electronic Commerce*. University of Kaiserslauten Computer Science Department, 1998.

F. Heister and W. Wilke. An architecture for maintaining case-based reasoning systems. In P. Cunningham, B. Smyth, and M. Keane, editors, *Proceedings of the Fourth European Workshop on Case-Based Reasoning*, Berlin, 1998. Springer Verlag.

D. Hennessy and D. Hinkle. Applying case-based reasoning to autoclave loading. *IEEE Expert*, 7(5):21–26, 1992.

E.N. Houstis, J.R. Rice, and R. Vichnevetsky, editors. *Expert Systems for Scientific Computing: Proceedings of the Second IMACS International Conference on Expert Systems for Numerical Computing*, Amsterdam, 1992. North-Holland.

E.N. Houstis, A. Catlin, J. Rice, V. Verykios, N. Ramakrishnan, and C. Houstis. PYTHIA II: A knowledge/database system for managing performance data and recommending scientific software. *ACM Transactions on Mathematical Software*, in press. http://www.cs.purdue.edu/research/cse/pythia/doc/pythia.pdf.

K. Hua and B. Faltings. Exploring case-based design - CADRE. *Artificial Intelligence in Engineering Design, Analysis and Manufacturing*, 7(2):135–144, 1993.

Neil Hurley. Evaluating the application of CBR in mesh design for simulation problems. In Manuela Veloso and Agnar Aamodt, editors, *Case-Based Reasoning Research and Development: Proceedings of the First International Conference on Case-Based Reasoning*, pages 193–204, Berlin, 1995. ICCBR, Springer Verlag.

L. Ihrig and S. Kambhampati. Storing and indexing plan derivations through explanation-based analysis of retrieval failures. *Journal of Artificial Intelligence Research*, 7:161–198, 1997.

CASUEL: A common case representation language. INRECA Consortium. Available on the World-Wide Web at http://wwwagr.informatik.uni-kl.de/~bergmann/casuel/CASUEL_toc2.04.fm.html, 1994.

International Organization for Standardization, Geneva, Switzerland. *ISO 8879: Information Processing – Text and Office Systems – Standard Generalized Markup Language (SGML)*, 1986.

M. Jaczynski and B. Trousse. WWW assisted browsing by reusing past navigations of a group of users. In P. Cunningham, B. Smyth, and M. Keane, editors, *Proceedings*

*of the Fourth European Workshop on Case-Based Reasoning*, pages 160–171, Berlin, 1998. Springer Verlag.

D. Jonassen, K. Beissner, and M. Yacci. *Explicit methods for conveying structural knowledge through concept maps*, chapter 15, page 155. Erlbaum, Hillsdale, NJ, 1993.

Anupam Joshi, S. Weerawarana, N. Ramakrishnan, E. Houstis, and J.R. Rice. Neuro-fuzzy support for problem solving environments: A step towards automated solution of PDEs. *IEEE Computational Science and Engineering*, 3(1):44–56, 1996.

Anupam Joshi, T. Drashanksy, J.R. Rice, S. Weerawarana, and E.N. Houstis. Multiagent simulation of complex heterogeneous models in scientific computing. *IMACS Math. and Comp. in Simulation*, 44:43–59, 1997.

M.S. Kamel, K.S. Ma, and W.H. Enright. ODEXPERT an expert system to select numerical solvers for initial value ODE systems. In Houstis et al. [1992], pages 33–54.

H. Kautz, editor. *Recommender Systems: Papers from the 1998 Workshop*. AAAI Press, Menlo Park, CA, 1998.

H. Kitano and H. Shimazu. The experience sharing architecture: A case study in corporate-wide case-based software quality control. In D. Leake, editor, *Case-Based Reasoning: Experiences, Lessons, and Future Directions*, pages 235–268. AAAI Press, Menlo Park, CA, 1996.

J. Kolodner and D. Leake. A tutorial introduction to case-based reasoning. In D. Leake, editor, *Case-Based Reasoning: Experiences, Lessons, and Future Directions*, pages 31–65. AAAI Press, Menlo Park, CA, 1996.

J. Kolodner. Improving human decision making through case-based decision aiding. *The AI Magazine*, 12(2):52–68, Summer 1991.

J. Kolodner. *Case-Based Reasoning*. Morgan Kaufmann, San Mateo, CA, 1993.

P. Koton. Smartplan: A case-based resource allocation and scheduling system. In K. Hammond, editor, *Proceedings of the DARPA Case-Based Reasoning Workshop*, pages 290–294, San Mateo, 1989. DARPA, Morgan Kaufmann.

M. Kriegsman and R. Barletta. Building a case-based help desk application. *IEEE Expert*, 8(6):18–26, December 1993.

T. Lane and C. Brodley. Approaches to online learning and concept drift for user identification in computer security. In *Papers from the AAAI 1998 Workshop on Predicting the Future: AI Approaches to Time-Series Problems*, pages 64–70. AAAI, 1998.

Patrick Laug. DOMINO: a knowledge-based system for the users of a finite element library. *Mathematics and Computers in Simulation*, 36(4–6):293–301, 1994.

D. Leake and D. Wilson. Case-base maintenance: Dimensions and directions. In P. Cunningham, B. Smyth, and M. Keane, editors, *Proceedings of the Fourth European Workshop on Case-Based Reasoning*, pages 196–207, Berlin, 1998. Springer Verlag.

D. Leake and D. Wilson. Combining CBR with interactive knowledge acquisition, manipulation and reuse. In *Proceedings of the Third International Conference on Case-Based Reasoning*, pages 203–217, Berlin, 1999. Springer Verlag.

D. Leake and D. Wilson. When experience is wrong: Examining CBR for changing tasks and environments. In *Proceedings of the Third International Conference on Case-Based Reasoning*, pages 218–232, Berlin, 1999. Springer Verlag.

D. Leake and D. Wilson. A case-based framework for interactive capture and reuse of design knowledge. *Applied Intelligence*, 14, 2000. In press.

D. Leake and D. Wilson. Remembering why to remember: Performance-guided case-base maintenance. In E. Blanzieri and L. Portinale, editors, *Proceedings of the Fifth European Workshop on Case-Based Reasoning*, pages 161–172, Berlin, 2000. Springer Verlag.

D. Leake, A. Kinley, and D. Wilson. Learning to improve case adaptation by introspective reasoning and CBR. In *Proceedings of the First International Conference on Case-Based Reasoning*, pages 229–240, Berlin, October 1995. Springer Verlag.

D. Leake, A. Kinley, and D. Wilson. Acquiring case adaptation knowledge: A hybrid approach. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 684–689, Menlo Park, CA, 1996. AAAI Press.

D. Leake, A. Kinley, and D. Wilson. Case-based CBR: Capturing and reusing reasoning about case adaptation. *International Journal of Expert Systems*, 10(2):197–213, 1997.

D. Leake, A. Kinley, and D. Wilson. A case study of case-based CBR. In *Proceedings of the Second International Conference on Case-Based Reasoning*, pages 371–382, Berlin, 1997. Springer Verlag.

D. Leake, A. Kinley, and D. Wilson. Learning to integrate multiple knowledge sources for case-based reasoning. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 246–251. Morgan Kaufmann, 1997.

D. Leake, L. Birnbaum, K. Hammond, C. Marlow, and H. Yang. Integrating information resources: A case study of engineering design support. In *Proceedings of the Third International Conference on Case-Based Reasoning*, pages 482–496, Berlin, 1999. Springer Verlag.

D. Leake. Towards a computer model of memory search strategy learning. In *Proceedings of the Sixteenth Annual Conference of the Cognitive Science Society*, pages 549–554, Hillsdale, NJ, 1994. Lawrence Erlbaum.

D. Leake, editor. *Case-Based Reasoning: Experiences, Lessons, and Future Directions*. AAAI Press/MIT Press, Menlo Park, CA, 1996.

D. Leake. Experience, introspection, and expertise: Learning to refine the case-based reasoning process. *The Journal of Experimental and Theoretical Artificial Intelligence*, 8(3), 1996.

M. Lenz and K. Ashley, editors. *Proceedings of the AAAI-98 workshop on textual case-based reasoning*. AAAI Press, Menlo Park, CA, 1998.

M. Lenz, B. Bartsch-Spörl, H.-D. Burkhard, and S. Wess, editors. *Case-Based Reasoning Technology: From Foundations to Applications*. Springer, Berlin, 1998.

Mario Lenz, Karl-Heinz Bush, André Hübner, and Stefan Wess. The SIMATIC knowledge manager. In *Proceedings of the AAAI-99 Workshop on Exploring Synergies of Knowledge Management and Case-Based Reasoning*, pages 40–45, 1999.

W. Mark, E. Simoudis, and D. Hinkle. Case-based reasoning: Expectations and results. In D. Leake, editor, *Case-Based Reasoning: Experiences, Lessons, and Future Directions*, pages 269–294. AAAI Press, Menlo Park, CA, 1996.

David McSherry. Intelligent case-authoring support in casemaker-2. *Computational Intelligence*, 17(2), 2001.

Mirjam Minor and Alexandre Hanft. Corporate knowledge editing with a life cycle model. In *Proceedings of the Eigth German Workshop on Case-Based Reasoning*, 2000.

H. Muñoz-Avila and J. Huellen. Feature weighting by explaining case-based planning episodes. In *Proceedings of the Third European Workshop on Case-Based Reasoning*, pages 280–294, Berlin, 1996. Springer Verlag.

H. Muñoz-Avila. A case retention policy based on detrimental retrieval. In *Proceedings of ICCBR-99*, 1999.

D. Navinchandra. Case-based reasoning in CYCLOPS, a design problem solver. In J. Kolodner, editor, *Proceedings of the DARPA Case-Based Reasoning Workshop*, pages 286–301, Palo Alto, 1988. DARPA, Morgan Kaufmann.

B. D. Netten. Verification of case-base integrity in BRIDGE. In *Proceedings of the Seventh German Workshop on Case-Based Reasoning*, pages 120–130, 1999.

J.D. Novak and D.B. Gowin. *Learning How to Learn*. Cambridge University Press, New York, 1984.

P. Pirolli and J. Anderson. The role of learning from examples in the acquisition of recursive programming skills. *Canadian Journal of Psychology*, 39:240–272, 1985.

L. Portinale, P. Torasso, and P. Tavano. Dynamic case memory management. In *Proceedings of the Thirteenth European Conference on Artificial Intelligence*, pages 73–77, Chichester, 1998. Wiley.

L. Portinale, P. Torasso, and P. Tavano. Speed-up, quality, and competence in multi-modal reasoning. In *Proceedings of the Third International Conference on Case-Based Reasoning*, pages 303–317, Berlin, 1999. Springer Verlag.

K. Racine and Q. Yang. Maintaining unstructured case bases. In *Proceedings of the Second International Conference on Case-Based Reasoning*, pages 553–564, Berlin, 1997. Springer Verlag.

Naren Ramakrishnan, John R. Rice, and Elias N. Houstis. GAUSS an online algorithm recommender system for one-dimensional numerical quadrature. *ACM Transactions on Mathematical Software*, 2000. To Appear.

Naren Ramakrishnan. *Recommender Systems for Problem Solving Environments.* PhD thesis, Purdue University, Lafayette, IN, 1997.

M. Redmond. *Learning by Observing and Understanding Expert Problem Solving.* PhD thesis, College of Computing, Georgia Institute of Technology, 1992. Technical report GIT-CC-92/43.

T. Reinartz, I. Iglezakis, and T. Roth-Berghofer. On quality measures for case base maintenance. In E. Blanzieri and L. Portinale, editors, *Proceedings of the Fifth European Workshop on Case-Based Reasoning*, Berlin, 2000. Springer Verlag.

M. Richter. Introduction. In M. Lenz, B. Bartsch-Spörl, H.-D. Burkhard, and S. Wess, editors, *CBR Technology: From Foundations to Applications*, chapter 1, pages 1–15. Springer, Berlin, 1998.

C. Riesbeck and R.C. Schank. *Inside Case-Based Reasoning.* Lawrence Erlbaum, Hillsdale, NJ, 1989.

C. Riesbeck. What next? The future of CBR in postmodern AI. In D. Leake, editor, *Case-Based Reasoning: Experiences, Lessons, and Future Directions.* AAAI Press, Menlo Park, CA, 1996.

B. Ross. Remindings and their effects in learning a cognitive skill. *Cognitive Psychology*, 16:371–416, 1984.

B. Ross. Some psychological results on case-based reasoning. In K. Hammond, editor, *Proceedings of the DARPA Case-Based Reasoning Workshop*, pages 144–147, San Mateo, 1989. Morgan Kaufmann.

M. Salganicoff. Tolerating concept and sampling shift in lazy learning using prediction error context switching. *Artificial Intelligence Review*, 11(1-5):133–155, 1997.

Gerard Salton and Chris Buckley. Term-weighting approaches in automatic text retrieval. *Information Processing and Management*, 24(5):513–523, 1988.

R.C. Schank. *Dynamic Memory: A Theory of Learning in Computers and People.* Cambridge University Press, Cambridge, England, 1982.

H. Schmidt, G. Norman, and H. Boshuizen. A cognitive perspective on medical expertise: Theory and implications. *Academic Medicine*, 65(10):611–621, 1990.

Arijit Sengupta. Toward the union of databases and document management: The design of DocBase. In *Proceedings of COMAD-98*. Tata McGraw Hill, 1998.

H. Shimazu and Y. Takashima. Detecting discontinuities in case-bases. In *Proceedings of the Thirteenth National Conference on Artifical Intelligence*, volume 1, pages 690–695, Menlo Park, CA, 1996. AAAI Press.

Hideo Shimazu, Hiroaki Kitano, and Akihiro Shibata. Retrieving cases from relational data-bases: Another stride towards corporate-wide case-base systems. In *Proceedings of IJCAI-93*, 1993.

Hideo Shimazu. A textual case-based reasoning system using XML on the world-wide web. In *Proceedings of the Fourth European Workshop on Case-Based Reasoning.* Springer, 1998.

S. Shiu, C. Sun, X. Wang, and D. Yeung. Maintaining case-based reasoning systems using fuzzy decision trees. In E. Blanzieri and L. Portinale, editors, *Proceedings of the Fifth European Workshop on Case-Based Reasoning*, Berlin, 2000. Springer Verlag.

S.C.K. Shiu, C.H. Sun, X.Z. Wang, and D.S. Yeung. Transferring case knowledge to adaptation knowledge: An approach for case-base maintenance. *Computational Intelligence*, 17(2), 2001.

E. Simoudis, K. Ford, and A. Cañas. Knowledge acquisition in case-based reasoning: "...and then a miracle happens". In D. Dankel, editor, *Proceedings of the 1992 Florida AI Research Symposium.* FLAIRS, 1992.

I. Smith, C. Lottaz, and B. Faltings. Spatial composition using cases: IDIOM. In *Proceedings of First International Conference on Case-Based Reasoning*, pages 88–97, Berlin, October 1995. Springer Verlag.

B. Smyth and P. Cunningham. The utility problem analysed: A case-based reasoning perspective. In *Proceedings of the Third European Workshop on Case-Based Reasoning*, pages 392–399, Berlin, 1996. Springer Verlag.

B. Smyth and M. Keane. Remembering to forget: A competence-preserving case deletion policy for case-based reasoning systems. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, pages 377–382, San Francisco, August 1995. Morgan Kaufmann.

B. Smyth and M. Keane. Design à la Déjà Vu: Reducing the adaptation overhead. In D. Leake, editor, *Case-Based Reasoning: Experiences, Lessons, and Future Directions*. AAAI Press, Menlo Park, CA, 1996.

B. Smyth and M. Keane. Adaptation-guided retrieval: Questioning the similarity assumption in reasoning. *Artificial Intelligence*, 102(2):249–293, 1998.

B. Smyth and E. McKenna. Modelling the competence of case-bases. In P. Cunningham, B. Smyth, and M. Keane, editors, *Proceedings of the Fourth European Workshop on Case-Based Reasoning*, pages 208–220, Berlin, 1998. Springer Verlag.

B. Smyth and E. McKenna. Building compact competent case-bases. In *Proceedings of the Third International Conference on Case-Based Reasoning*, Berlin, 1999. Springer Verlag.

B. Smyth and E. McKenna. Footprint-based retrieval. In *Proceedings of the Third International Conference on Case-Based Reasoning*, Berlin, 1999. Springer Verlag.

B. Smyth and E. McKenna. An efficient and effective procedure for updating a competence model for case-based reasoners. In *Proceedings of the Eleventh European Conference on Machine Learning*, Berlin, 2000. Springer Verlag.

Barry Smyth and Elizabeth McKenna. Competence models and the maintenance problem. *Computational Intelligence*, 17(2), 2001.

B. Smyth. Case-base maintenance. In *Proceedings of the Eleventh International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, Castellon, Spain, 1998.

Markus Stolpmann and Stefan Wess. *Intelligente Systeme für E-Commerce und Support*. Addison Wesley, 1998.

E. Stroulia, M. Shankar, A. Goel, and L. Penberthy. A model-based approach to blame assignment in design. In J. Gero, editor, *Artificial Intelligence in Design*, pages 519–537, Boston, 1992. Kluwer.

Jerzy Surma and Janusz Tyburcy. A study on competence-preserving case replacing strategies in case-based reasoning. In P. Cunningham, B. Smyth, and M. Keane, editors, *Proceedings of the Fourth European Workshop on Case-Based Reasoning*, pages 233–238. Springer Verlag, 1998.

K. Sycara and D. Navinchandra. Index transformation and generation for case retrieval. In K. Hammond, editor, *Proceedings of the DARPA Case-Based Reasoning Workshop*, pages 324–328, San Mateo, 1989. DARPA, Morgan Kaufmann.

K. Sycara, R. Guttal, J. Koning, S. Narasimhan, and D. Navinchandra. CADET: a case-based synthesis tool for engineering design. *International Journal of Expert Systems*, 4(2):157–188, 1991.

K. Sycara. *Resolving Adversarial Conflicts: An Approach Integrating Case-based and Analytic Methods*. PhD thesis, School of Information and Computer Science, Georgia Institute of Technology, 1987. Georgia Institute of Technology, Technical Report GIT-ICS-87/26.

Houman Talebzadeh, Sanda Mandutianu, and Christian Winner. Countrywide loan-underwriting expert system. *AI Magazine*, 16(1):51–64, 1995.

M. van Someren, J. Surma, and P. Torasso. A utility-based approach to learning in a mixed case-based and model-based reasoning architecture. In *Proceedings of the Second International Conference on Case-Based Reasoning*, pages 477–488, Berlin, 1997. Springer Verlag.

M. Veloso, A. Mulvehill, and M. Cox. Rationale-supported mixed-initiative case-based planning. In *Proceedings of the Ninth Conference on Innovative Applications of Artificial Intelligence*, pages 1072–1077, Menlo Park, CA, 1997. AAAI Press.

M. Veloso. *Planning and Learning by Analogical Reasoning*. Springer Verlag, Berlin, 1994.

Juan Villacis, Madhusudhan Govindaraju, David Stern, Andrew Whitaker, Fabian Breg, Prafulla Deuskar, Benjamin Temko, Dennis Gannon, and Randall Bramley. CAT: A high performance distributed component architecture toolkit for the grid. In *Proceedings of the High Performance Distributed Computing Conference*, 1999.

Ivo Vollrath, Wolfgang Wilke, and Ralph Bergmann. Case-based reasoning support for online catalog sales. *IEEE Internet Computing*, 2(4), 1998.

I. Vollrath. Reuse of complex electronic designs: Requirements analysis for a CBR application. In P. Cunningham, B. Smyth, and M. Keane, editors, *Proceedings of the Fourth European Workshop on Case-Based Reasoning*, pages 136–147, Berlin, 1998. Springer Verlag.

A. Voß. The need for knowledge acquisition in case-based reasoning – some experiences from an architectural domain. In *Proceedings of the Eleventh European Conference on Artificial Intelligence*, pages 463–467. John Wiley, 1994.

Angi Voß. Principles of case reusing systems. In I. Smith and B. Faltings, editors, *Advances in case-based reasoning*, pages 428–444, Berlin, 1996. Springer Verlag.

W3C. *XML-QL: A query language for XML*, 1998. http://www.w3.org/TR/1998/NOTE-xml-ql-19980819/.

Ian Watson and Dan Gardingen. A distributed case-based reasoning application for engineering sales support. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, volume 1, pages 600–605. Morgan Kaufmann, 1999.

I. Watson. *Applying Case-Based Reasoning: Techniques for Enterprise Systems*. Morgan Kaufmann, San Mateo, CA, 1997.

S. Weerawarana, E. Houstis, J.R. Rice, Anupam Joshi, and C. Houstis. PYTHIA: A knowledge based system to select scientific algorithms. *ACM Trans. Mathematical Software*, 22(4):447–468, 1997.

D. Wettschereck, D. Aha, and T. Mohri. A review and empirical evaluation of feature-weighting methods for a class of lazy learning algorithms. *Artificial Intelligence Review*, 11(1-5):273–314, February 1997.

W. Wilke, I. Vollrath, K.-D. Althoff, and R. Bergmann. A framework for learning adaptation knowedge based on knowledge light approaches. In *Proceedings of the Fifth German Workshop on Case-Based Reasoning*, pages 235–242, 1997.

David C. Wilson and David B. Leake. Maintaining case-based reasoners: Dimensions and directions. *Computational Intelligence*, 17(2), 2001.

Qiang Yang and Jing Wu. Keep it simple: A case base maintenance policy based on clustering and information theory. In *Proceedings of the Canadian AI Conference*, May 2000.

Zhong Zhang and Qiang Yang. Towards lifetime maintenance of case base indexes for continual case based reasoning. In *Proceedings of the 1998 International Conference on AI Methodologies, Systems and Applications (AIMSA-98)*, pages 489–500, Berlin, 1998. Springer Verlag.

Zhong Zhang and Qiang Yang. Dynamic refinement of feature weights using quantitative introspective learning. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*. Morgan Kaufmann, 1999.

Jun Zhu and Qiang Yang. Remembering to add: Competence-preserving case-addition policies for case base maintenance. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*. Morgan Kaufmann, 1999.