Université de Montréal

A Case-Based Reasoning Diagnosis System for AHU (Air-Handling Unit)

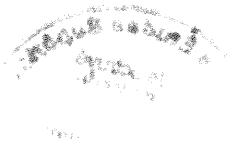
Par

Suoshi Zheng

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Mémoire présenté à la Faculté des études supérieures
en vue de l'obtention du grade de
Maître ès sciences (M.Sc.)
en informatique

Août, 2002



Université de Montréal

Faculté des etudes supérieures

Ce mémoire intitulé :

A Case-Based Reasoning Diagnosis System
for AHU (Air-Handling Unit)

présenté par:

Suoshi Zheng

a été évalué par un jury composé des personnes suivantes :

président-rapporteur: Philippe Langlais

directeur de recherche: Houari Sahraoui

membre du jury: Jian -Yun Nie

Mémoire accepté le:
4 Novembre 2002

Résumé

A Case-Base Reasoning Diagnosis System for AHU (Air-Handling Unit)

Suoshi Zheng

Le directeur: Houari Sahraoui

Récemment, les systèmes de raisonnement à base de cas (CBR) sont devenus de plus en plus populaires pour résoudre une large variété de problèmes à connaissance intensive. Cette thèse se propose de développer un système de diagnostic de défauts pour une unité de traitement d'air (AHU) en utilisant une approche de CBR.

L'objectif principal du AHU est de fournir des conditions d'air appropriées pour les bâtiments commerciaux. Le diagnostic des défauts est une tâche permettant d'identifier les causes de tout disfonctionnement dans le AHU. Si toutes les pannes peuvent être localisées rapidement et avec précision, alors il sera possible d'éviter une consommation excessive d'énergie ou un inconfort des occupants.

CBR est une application à base de connaissance. En effet, CBR est basé sur le principe d'utiliser les expériences précédentes comme référence pour résoudre les nouveaux problèmes qui se présentent dans des conditions pareilles. Les apports majeurs de l'application de CBR comme un moyen pour acquérir la faculté du diagnostic sont discutés. Ce mémoire est aussi concerné par l'intégration des principaux processus CBR dans un cadre d'application indépendant du domaine. L'ultime objectif de ce cadre d'application est de faciliter le développement de futures applications CBR par la réutilisation des conceptions et des implémentations.

Mot clés : base de cas, base de connaissance, unité de traitement d'air

Abstract

A Case-Based Reasoning Diagnosis System for AHU (Air-Handling Unit)

Suoshi Zheng

Supervisor: Houari Sahraoui

In recent years, case-based reasoning (CBR) systems have become increasingly popular as a way to solve a wide variety of knowledge intensive problems. This thesis addresses the development of a fault diagnosis system for an air-handling unit (AHU) by using a CBR approach.

The main aim of AHU is to deliver an appropriate air condition in commercial buildings. Fault diagnosis is the task of identifying the causes of all kinds of failures in an AHU. If all faults in AHU can be localized rapidly and accurately, this means that it will be able to avoid excessive energy consumption and discomforts to human occupants.

CBR is a sort of knowledge-based applications. CBR is based on the idea of using previous experiences as references to solve new problems that have similar situations with the previous experiences. The potential benefits of applying CBR to enhance the diagnostic capability are discussed. This thesis is also concerned with integrating main CBR processes into a domain-independent framework. The greatest desire for this framework is to facilitate the development of later CBR applications by reusing the previous designs and implementations.

Key words: case-based reasoning, knowledge-based system, framework, and air-handling unit.

Table of Contents

List of Figures

List of Tables

Dedication

To my wonderful parents and lovely wife

Acknowledgements

This thesis is the fruition of a lot of effort, enthusiasm and hard work. I would like to thank many people for their kindness and assistance during my research work. First of all, I sincerely thank my supervisor, Professor Houari Sahraoui, who provided me the opportunity to work for a real world problem and who advised me in every step of my research. I would not have finished this thesis without his support and advice.

I would like to thank other members of dissertation committee, Professors Philippe Langlais and Jian -Yun Nie, who devoted them time to review this thesis, and provided me with valuable feedback and comments.

Many thanks to the team of this project, especially the project assistant, Siveton Vincent, a funny French guy, who speaks French like a machine gun. With my poor French, sometimes, we had to involve a lot of body languages to communicate, but we never failed to make quality discussion during the development of this project. I also thank Dr. David Grosser, whose experience in knowledge process is so impressive.

I truly appreciate the support of Energy Diversification Research Laboratory (CEDRL). Mr. Daniel Choiniere helped me to understand the domain, and Mrs. Maria Corsi provided me a lot of history data and explanations.

Most importantly, I am grateful to my parents. They have always been there for standing by me. Without their warm encouragement and love, my schoolwork would have been much harder.

Finally, I thank my wife, Hang Qin, for her understanding and patience. Her harmless yelling and love have always given me the strength and motivation throughout my career.

Chapter 1.

Introduction

1.1. Overview

It has been often said that knowledge is power. This expression, in my opinion, carries at least two different meanings. The first is that knowledge can be applied to solve a new problem if we have had some relevant experiences stored in knowledge. The second is that we should learn new knowledge constantly in order to get ready for any potential problems in the future. There is no knowledge that is no power. This is especially true in the development of knowledge-based systems.

Knowledge-based systems were initially developed in the artificial intelligence (AI) community. A knowledge-based system is a computer system that includes a knowledge base used to capture the essential features of a domain problem and makes that information accessible to a problem solving procedure. The central component of knowledge-based systems is its knowledge base, which is an organizational data structure based on computer technologies to represent

particular aspects of the knowledge that we are concerned with in a domain. The knowledge in a knowledge base can be exhibited in either general or specific manners. The general knowledge usually refers to the higher-level abstractions or generalizations of universal principles in a given domain; the specific knowledge refers to detailed situations about individual objects or events. In contrast with conventional computer systems, knowledge-based systems were born to help in knowledge 'storages' and 'reuses'. The capabilities of expressing and processing knowledge in knowledge-based systems are able to significantly increase the intelligence level of computer systems and achieve better success in problem solving.

Knowledge-based systems run through a phase from simple collections of frequently asked questions (FAQ) to complex systems in terms of AI engines. Since the introduction of the first knowledge-based system in 1970s, these kinds of systems have made a number of success stories, especially in rule-based expert systems. For a long time, AI research focused on rule-based expert systems that used heuristic if-then rules to represent domain knowledge and applied an inference procedure to manipulate those rules in order to reach a conclusion. Rule-based expert systems can advise on, or help solve, a real problem as though the end-users were dealing with a human expert in the particular domain. However, despite the undoubted success of rule-based expert systems, there are still some well-known problems in developing such systems, including knowledge elicitation and maintaining issues [1]. In recent years, the trend of research shifted from this rule-based approach to a new approach that is case-based reasoning (CBR).

In this thesis, a close study of using CBR to solve a practical domain problem is presented; namely, a CBR diagnosis system for an air-handling unit (AHU). The basic idea of CBR is to use previous experiences as references in solving new problems that have similar situations to the previous experiences. The key

assumption is that if two problems look alike, then the solutions to these problems would be highly and frequently close as well.

1.2. Problem Statement

Heating ventilating and air-conditioning (HVAC) systems have become absolutely necessary in modern buildings. HVAC systems are certain kinds of engineered systems that offer human occupants an acceptable indoor air condition in the building space. The quality of the indoor air is of great importance for the health and comfort of human occupants. Nevertheless, since there is a growing issue concerning the Global Energy Crisis, the energy used by HVAC systems has received greater attention among the total energy consumption in the world. The systems with higher reliability and lower running costs are highly demanded.

An HVAC system is typically composed of an air-handling unit (AHU) and several variable air volume (VAV) boxes. As a central element of an HVAC system, the role of AHU is to provide high quality air condition relating to appropriated air temperature, proper humidity level and adequate ventilation in the entire building space. Normally, an AHU consists of outdoor, mixing and return air dampers, cooling and heating coils, an air filter section, supply and return fans, and an humidifier. AHU systems are responsible for a significant portion of the total building energy consumption and a major assurance of comfort conditions in the entire building space.

However, with the dramatic developments in computer auto-controlled technologies, AHU systems have become more complex than ever. Large numbers of sensors, control equipment and communication units have been integrated into AHU. Due to the complexity of AHU and the long period of running time, faults occurring in AHU are common, such as sensor drift, stuck damper or fan failure. All kinds of failures can lead to excessive energy

consumption, discomforts to human occupants, and increasing premature wear on control equipment.

To solve these problems, there is a need for automatic and robust fault detection and diagnosis (FDD) tools. Fault detection is a task of determining that the operation of the building is incorrect or unacceptable in some respect, whereas, fault diagnosis is a task of identifying or localizing of the causes of failures by using situation descriptions and behavior characteristics [2]. Fault diagnosis is more difficult than fault detection, because fault diagnosis usually requires a vast amount of knowledge to understand the principles of how all elements in AHU interact each other.

The objective of this thesis is to develop a prototype of diagnostic method and tool for AHU, which hopefully can lead to a commercial diagnostic product. The research is part of the work in the project of Fault Detection and Diagnosis Tool – Stand Alone and Embedded Application - which is sponsored by the Energy Diversification Research Laboratory (CEDRL) [3] and Delta Controls Company [4]. The major research interest of CEDRL and Delta Controls are emphasis on establishing intelligent energy management and control systems with lower operating costs. Moreover, this thesis is also concerned with integrating main CBR processes into a domain-independent framework. The greatest desire for this framework is to facilitate the development of later CBR applications by reusing the previous designs and implementations.

1.3. Scope of Thesis

This thesis discusses the use of CBR techniques to develop a diagnosis system for AHU. The thesis is focused on four following topics:

- CBR and related research work

- Key Issues in CBR processing
- Fault detection and diagnosis in AHU
- CBR diagnosis system

1.3.1. CBR and Related Research Work

In order to enhance the capability of computer systems in complex problem solving, one attempt is to build knowledge-based systems. A knowledge-based system is a computer system that encodes sufficient knowledge into computers and applies a certain inference mechanism to produce intelligent behaviors. In the first topic of this thesis, typical knowledge-based systems are introduced, including CBR systems and rule-based expert systems. The critical issues in developing knowledge-based systems are reflected in the following questions: How the explicit knowledge can be attained from a real domain problem? How this knowledge can be organized into a computer-understandable form to make the reasoning process possible? How machine learning can be performed as new knowledge comes along? To answer these questions, knowledge acquisition issue and knowledge representation structures are explained. Through the discussion of strengths and weaknesses in CBR and rule-based approaches, a conclusion is given that CBR is a considerable methodology to address domain problems with a weak theory [5].

1.3.2. Key Issues in CBR processing

In this topic, a comprehensive overview of CBR techniques is introduced. It starts by explaining the philosophy of CBR, which shows that CBR not only is a powerful methodology for computer reasoning, but also forms a common behavior of humans to solve problems in their daily life. To understand a little bit more of what CBR essentially does, the CBR cycle is explained. The CBR cycle

can be looked at as a guideline to develop CBR applications. Some sub-fields relating to this thesis in the development of CBR applications are discussed, such as case representation, case retrieval, case adaptation, and learning mechanisms.

- The case representation problem concerns how to express general and specific domain knowledge to the case base.

- The case retrieval problem concerns how to efficiently retrieve cases that are the most similar to the current problem from the case base.

- The case adaptation problem concerns how to modify a retrieved case to make it better fit the current problem situation.

- The learning mechanism problem concerns how to update the case base to keep up with an evolving environment.

1.3.3. Fault Detection and Diagnosis (FDD) in AHU

The domain problem concerning AHU is addressed here. The functionality of each single element in AHU is described. AHU systems are responsible for delivering an appropriated air condition in the entire building space. The purpose of the FDD scheme is explained. The execution of the FDD task relies highly on how to define an FDD scheme to understand system behaviors on what is expected and what is wrong, since the raw information of each single element in AHU does not suffice to determine whether the system runs under normal or abnormal situations.

As mentioned above, the task of FDD is comprised of two essential subtasks (fault detection and diagnosis). The fault detection is a subtask of revealing all possible physical failures in AHU during running time. To detect all possible faults, a rule-based expert system has already been developed successfully by

CEDRL. The fault diagnosis is a subtask of localizing the causes. However, the diagnostic task in AHU is more complicated than the detective task. A rule-based expert system is no longer suitable for the diagnostic task and is likely to produce incomplete or inaccurate results. The reasons that we proposed to use CBR to achieve the diagnostic task can be summarized below:

- The dynamic behavior of AHU is often non-linear and poorly understood. It is very difficult to obtain adequate representation of the complex behavior of AHU using rule-based approach; on the contrary, CBR approach is often suitable for where rule-based systems find it hard to generate the rules from poor theoretical domains.

- The performance of rule-based expert systems does not satisfy as well as indented for the diagnostic task in AHU; one of the reasons is that critical information is lost after rules are formed. In AHU, all elements take affect in a single duct so that the control behavior can be heavily influenced each other, thus, it is required to look over entire system to determine a fault. In CBR, a case is a snapshot of the situation of all elements at a single moment. This snapshot can be analyzed to diagnose any faults in AHU.

- There is only limited causal information for why elements fail in AHU, and different faults might have similar symptoms. Rule-based expert systems are not able to identify the fault unambiguously. A CBR diagnosis system is able to compare each actual situation in detail so that the real causes behind similar symptoms can be figured out.

- Lastly, there is plenty of time-series data produced by building energy management systems (BEMS) during the monitoring period. All these data can be easily transformed into cases.

As a result, to completely overcome the FDD problem, a rule-based detection system and a CBR diagnosis system are blended in this project. The integration of two knowledge-based systems is able to significantly enhance the capability of the FDD in AHU.

1.3.4. CBR Diagnosis System

The explanation of our approach to develop a CBR diagnosis system for AHU is given here in detail. The major intentions of this thesis are not limited to the development of a prototype of the CBR diagnosis system, but are also concerned in constructing an object-oriented diagnostic framework for various kinds of domains. The basic idea is the development of a reusable and extensible environment that integrates a domain-independent case representation model with certain generic reasoning methods. The code and design reuse capability of object-oriented frameworks enables higher productivity and a shorter time-to-market of application development when compared with traditional software systems development [6].

This thesis took a different approach to develop a diagnostics problem by using CBR methodology. Using the CBR approach to solve the diagnostic problem and develop the generic framework presented us with some of the following challenges:

- How to structure and represent the domain knowledge (AHU)?
- How to map over the gap between domain-specific and domain-independent knowledge during the design of a generic CBR framework?
- How to assess the similarity between the two cases?
- How to achieve learning?

In order to meet these challenges mentioned above, in our approach, we combined typical CBR techniques with an object-oriented case representation and an open structure of similarity measure. The diagnostic problem is accomplished by solving the following problems:

- Object-oriented case representation.
- Case retrieval.
- Evaluating and learning mechanism.

Case representation: In a CBR diagnosis system, each case mainly contains a problem description, which describes features of the state of AHU when the case occurred, and a solution, which describes the derived solution to the problem. In this thesis, we present an object-oriented case representing structure in a CBR framework to model the domain knowledge into object-oriented concepts. The case representation in our approach is divided into three-layers. The top layer is the package of Case-Base, which emphasizes the conceptual structure of the case base rather than being domain specific. The package of Domain-Object in the bottom layer is used to represent individual domain concepts in AHU. In the middle layer, we defined a package called Object-Network, which denotes the relations between the cases in the case base and domain objects in AHU for building a completed case.

Case retrieval: This concerns how to find the cases that are the most similar ones to the current problem in the case base. A k-nearest neighbor algorithm is primarily selected to integrate into the open structure of similarity calculations. The k-nearest neighbor is perhaps the most widely used algorithm to calculate the similarity in CBR. It has shown to be very effective for a variety of domains. The

functions of similarity measure could be domain-independent in the CBR framework, and it still keeps open to add new reasoning methods later on.

Evaluating and learning mechanism: New solutions should be confirmed or validated by human experts. Human experts are able to modify the solution case if any change is necessary. The learning mechanism is accomplished by appending a new case to the case base.

There are several advantages in using CBR to solve the diagnostic problem for AHU. First of all, we want to benefit from CBR to reduce the need to acquire explicit models of problem domains. Due to the complexity of AHU systems, sometimes, it is still difficult to get a clear idea of the interaction among all the independent elements in AHU. Secondly, CBR systems can also be an increasing learning process by acquiring new cases. Once a solution is found, it can be inserted into the case base as a new case, so that the performances of CBR systems will be improved with time. Lastly, the solutions suggested by CBR systems may be more accurate, because each case reflects what really happened under a particular circumstance in the past.

1.4. Organization of Thesis

The rest of the thesis is organized into the following chapters:

- Chapter 2 introduces typical knowledge-based systems, including CBR systems and rule-based expert systems. The advantages and disadvantages of each approach in problem solving are given.

- Chapter 3 generally describes the CBR methodology in problem solving. The major sub-fields of the CBR process are explained.

- Chapter 4 gives a discussion of the domain problem concerning AHU and the diagnostic scheme. The architecture of the FDD tool is introduced.

- Chapter 5 presents our approach by using CBR to realize the fault diagnosis system for AHU, and a conceptual CBR framework is illustrated.

- Chapter 6 explains the implementation of a CBR diagnosis system and exhibits the result of experiment and testing.

- Chapter 7 gives a final conclusion and talks about possible future work.

Chapter 2.

CBR and Related Research Work

2.1. Overview

For over a half-century, in the AI community, there have been several approaches that attempt to bring intelligence into computers in order to endow computers with the same kind of flexibility as that of humans in problem solving. One of these approaches is the study of building knowledge-based systems. The target of the development of knowledge-based systems is to emulate the high-level skills of humans in a program running on computers. Informally, a knowledge-based system is a computer system that encodes sufficient knowledge into computers and applies a certain inference mechanism to produce intelligent behaviors. Typical knowledge-based systems include CBR systems and rule-based expert systems. In this chapter, the typical knowledge-based systems and relevant research works are introduced. Section 2.2 discusses the basis structure of knowledge-based systems. Section 2.3 explains the major issues of building a knowledge-based system, including knowledge acquisition and representation. In section 2.4 and 2.5, CBR and rule-based systems are introduced. Section 2.6

states the strengths and weaknesses of CBR and rule-based expert systems, respectively.

2.2. Basic Structure of Knowledge-based Systems

Nowadays, with the significantly developing success in computer and information technologies, people increasingly depend on computers to achieve 'mission critical' tasks in their daily work. Computers are fundamentally well suited for performing mechanical computing by using conventional programs. It is more efficient and reliable for computers to perform simple monotonous tasks than humans. However, for more complex problems, unlike humans – since the lack of the mechanism of representing and reasoning knowledge, computers are not able to understand specific situations and fit into new situations automatically; as a result, the performance of computers in complex problem solving is limited. Research on how to bring intelligence into computers to produce intelligent behaviors has led to many different approaches in AI. One of these is the research of building knowledge-based systems. Knowledge-based systems were born to help in utilization of computer and information technologies to store and reuse knowledge in order to make computers more useful. Actually, much of the inspiration for building such systems came from the desire to have the same kind of ability of reusing and learning knowledge as that of humans.

A knowledge-based system is concerned with representing and reasoning knowledge. The basic structure of knowledge-based systems is shown in Figure 2-1. The most important component in knowledge-based systems is its knowledge base, which is the place to provide the permanent storage of knowledge. By investigating the particular domain, explicit knowledge is extracted from either domain specifications or human experts and formulates into a structured knowledge base, which refers to something that we become aware of in the domain. However, it is not possible and necessary for the knowledge base to cover all aspects of the domain; some things can be treated as implicit knowledge that we remain unaware of in the domain.

Besides, knowledge-based systems not only serve for the storage of knowledge, but also include the ability to use its stored knowledge in explaining what is happening. Attached to this knowledge base are a reasoning engine and a user interface. The reasoning engine is used to deduce the implicit knowledge in terms of the explicit knowledge, if the problem is not directly contained in the knowledge base. The tasks of user interfaces are interacting with end-users, accepting and formulating problems, and providing an explanation of why and how the system arrived at a specific conclusion.

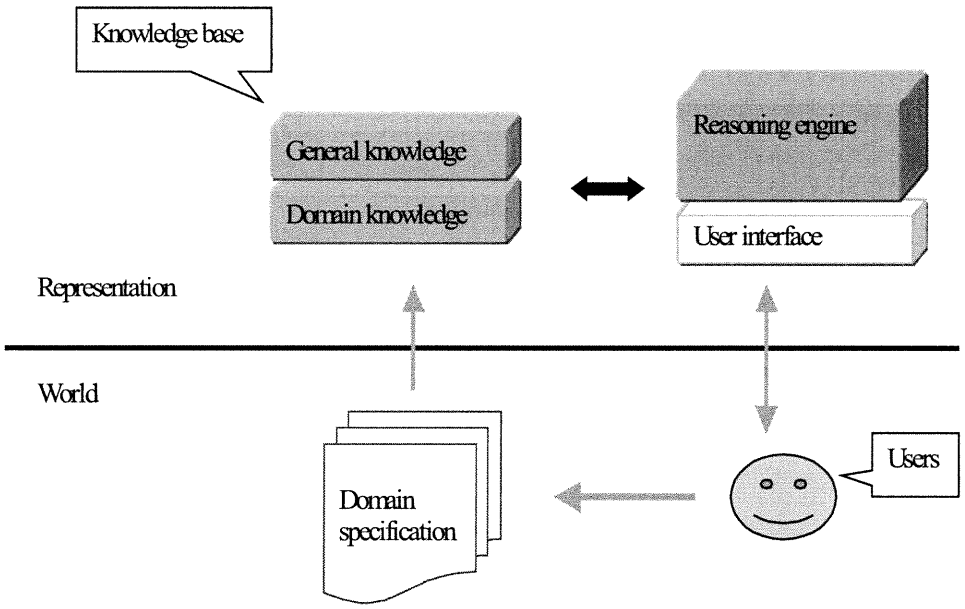


Figure 2 - 1 Knowledge-based system

Unlike conventional programming in which processes focus on manipulating data without understanding the underlying meaning of the data, knowledge-based systems concentrate on interpreting the data and their relationships. In addition, in conventional programming, such as scientific calculation, database programming and network computing, since the processes are defined by strict step-by-step instructions in programs, the results are predictable and certain. In knowledge-based systems, its results are a direct reflection of the amount of the knowledge

provided in the knowledge base. The conclusions would be satisfied if they were 'good enough' instead of a 'hundred percent correct'. [7]

Essentially, the performance of knowledge-based systems directly depends on the quality and quantity of its encoded knowledge in the knowledge base. If there is no adequate knowledge, the capabilities of problem solving are limited; however, knowledge stored in a knowledge base should be in a well organized structure to make a reasoning process easy to be accessed, managed and modified, otherwise, the systems can become inefficient, and even useless.

2.3. Building Knowledge-based Systems

Building and developing knowledge-based systems is made up of two main stages each having its functions:

- **Knowledge acquisition**, which is the most important phase of implementation in a knowledge-based system, as it involves how to gain explicit knowledge from domain specifications or human experts.

- **Knowledge representation**, in where the knowledge is organized as a set of computerized formulations or reasoning structures that reflect the variety of all domain concepts. A reasoning engine can draw certain decisions based on the knowledge representation layer.

Generally speaking, the process of knowledge acquisition gets ahead of knowledge representation design, since we need to learn about the domain before representing it.

2.3.1. Knowledge Acquisition

Knowledge acquisition is a process of gathering the explicit knowledge from any resource available in domains. What we obtained will later be used to create and develop a reasoning mechanism in the knowledge base. Knowledge acquistion is a difficult process that often requires special skills and takes much time. The work has to be done gradually, and normally consisits of gathering data, validating it, refining it, and then repeating the entire cycle [8].

During this process, two major tasks need to be accomplished: one is obtaining the knowledge from clients' requirements and another is defining concepts to represent the knowledge.

- The first task is concerned with identifying domain problem characteristics and understanding the reqirements from clients. It can be fulfilled either by giving interviews to domain experts or by studying the specification documents of the domain. One of the difficulties in this task is that we have to integrate domain knowledge from multiple resources, especially when inconsistencies and conflicts need to be resolved.

- The second task here is concerned with defining the concepts of the domain problem and clarifying the relations among those concepts. There are two different approachs to this task: one approach is an attemp at establishing an explict model of the domain, which includes the system relying on heuristic, causal, statistical, mathematical models. However, a huge challenge for this kind of systems is the requirement of generalizing a bunch of reliable principles and formulations from domain problems. In many domains, sometimes the principles and formulations may be either impossible to produce or too large to manage. This problem is well known to be a critical bottleneck for building such a knowledge-based system. In order to avoid this kind of 'dirty' job, a second approach is the research of building a system tied to specific knowledge directly as experience to

solve the domain problem. it might hopefully decrease the overload burden of knowledge acquisition issues by using specific experience instead of generalized formulations.

2.3.2. Knowledge Representation

The step after obtaining explicition knowledge is the process that decides how the knowledge about the domain can be translated into a computer language and stored in a form that makes it possble to be accessed by a reasoning engine. This is called the knowledge representation problem. The aims of knowledge representation is to study how knowledge can be organized, what kinds of reasoning can be done with its knowledge and how can learning be performed as new knowledge comes along.

The design of knowledge representation consists of three main issues with which we should be concerned:

- **Knowledge base,** where the knowledge is stored. Useful information is extracted from a specific domain into a structured knowledge base. The knowledge base can be looked at as mapping between the objects and relations in the real word and the computational objects and relations in the computer [9].

- **A reasoning engine,** which is the exposition of evidence in order to arrive at new conclusions. A reasoning engine usually consists of search and reasoning procedures that enable a knowledge-based system to reach solutions, and if required, provide justifications for its results. The implementation of a reasoning engine deeply depends on the knowledge base. Therefore, the way of reasoning engines that we design corresponding to the way of form or structure that we use to represent the knowledge.

- **A learning mechanism,** A widely shared view is that learning is a process to integrate new knowledge to a knowledge base or discard useless knowledge from a knowledge base. In order to meet the challenge of solving a real problem, knowledge-based systems must operate within an evolving environment, and their knowledge therefore needs continuous updating and refinement to make it potentially useful for later problems.

Over the past several decades, many different structures of knowledge representation corresponding to reasoning methods were proposed, such as case-based reasoning and rule-based representation.

2.4. Case-Based Reasoning

The CBR schemes represent knowledge as a library of cases (case base). Each of them records a particular situation of previous experiences and relevant solutions. The cases can be organized in a flat or hierarchical structure. The central tasks of CBR for reasoning shown in Figure 2-2 are to identify new problem descriptions, find a similar previous case to the new problem, and reuse or adapt the solution from the selected case to the new problem. Using CBR to solve a new problem relies on an assumption that a similar problem has been experienced in the past, and a solution to the problem has been saved. If it fails to find an appropriate solution, CBR systems are able to learn from this new experience and add a new case into the case base.

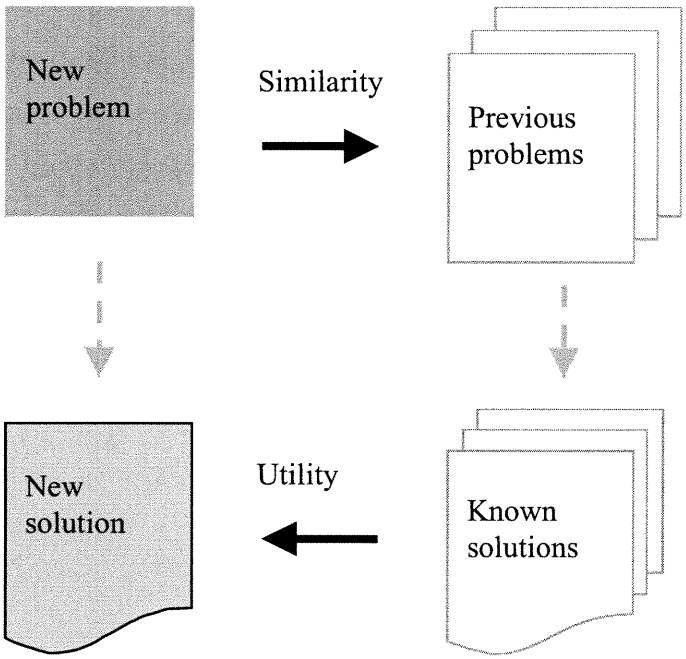


Figure 2 - 2 CBR approach

An example case in CBR is shown as below:

Case No. *11*

Problem descriptions:

Supply Air Temperature = 18 ℃

Supply Air Temperature Set Pont = 22 ℃

Heating Coil Valve Open = 100%

…

Outdoor Damper Open = 0

Solution:

Heating Coil Valve is stuck

2.5. Rule-Based Representation

The rule-based schemes consist of a large number of rules and objects. The rules are collections of "IF – THEN ---" conditions. The premise specifies certain patterns and the conclusion may be an action or assertion. The objects are a set of attributes that describe items of interest. One object is related to another object by symbolic links through the " IS---A--" mechanism. The rules in a knowledge base are not a simple map of knowledge, but instead go by abstracting from a domain specification or a human expert into a generalized way. The rules allow the system to deduce new results from an initial set of premises.

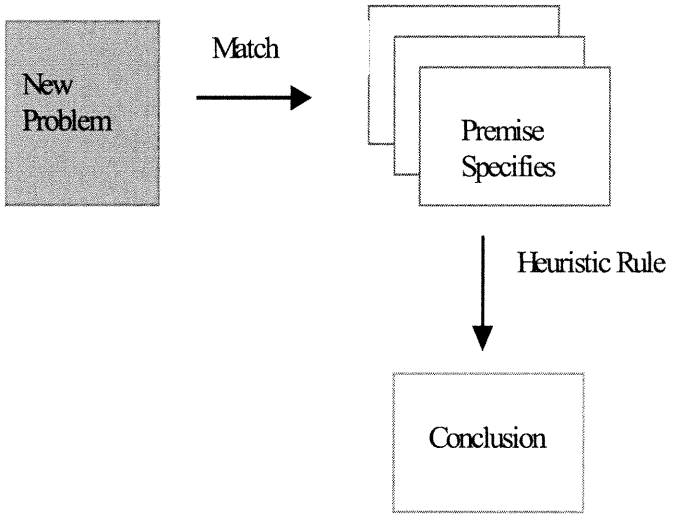


Figure 2 - 3 Rule-based approach

An example of rule is basically shown as following code:

IF

(Supply Air Temperature) is {less than Supply Air Set point}

AND

(Heating Coil Valve) is {open 100%}

THEN

(Mixing Damper) is {failure}

OR

(Outdoor Damper) is {failure}

OR

(Heating Coil Valve) is {stuck}

The reasoning methods run by tracing a path through the rules to reach the goals, with typically following one of the top-level strategies: forward chain or backward chain [10]. The forward chaining begins by gathering as much information as it can, then steps through the rule-base looking for the rules that may be satisfied by the information already gathered. Backward chaining works from the opposite direction, it starts with a goal and then tries to find the facts to support it. The rules are usually extracted from domains and encoded in the equation manually. Therefore, there is no generic method to perform learning in rule-based expert systems. As a result, the design of the learning mechanism in a rule-based system has to be domain-dependent so far.

2.6. CBR & Rule-Based Expert Systems

The idea behind developing different representing schemes is to attempt to make knowledge reasoning more efficiently. It does not mean that one representing scheme is absolutely better than others. Each of them has own their strengths and weaknesses according to what kind of problems are to be faced. Problem solving

in the real world involves different types of domain problems, such as open problems, strong theory problems, and complete theory problems [11].

An open problem is characterized by having a weak or intractable domain theory. A weak domain theory is applicable where the background knowledge existing for a particular domain is not powerful enough to completely describe all of the phenomena in the domain. In particular, such incomplete background knowledge may be too limited to allow the development of correct solutions for all of the possible problems that may arise in the domain. A strong problem is described by having more certain relationships between its concepts. A complete problem refers to the background knowledge existing for a particular domain that is able to cover all of the phenomena and relationships, and its facts can be expressed by either true or false. However, the real world is too complex to find a complete problem.

Rule-based expert systems are one of the success stories of AI research. They have made a great achievement in real problem solving, including strong theory problems, and sometimes open problems. Formulating knowledge in abstracted or generalized ways enables the rules in knowledge bases to have wide applicability; the rules could even be applied to different domains.

However, despite the undoubted success of the rule-based expert system in many domains, developing a rule-based expert system has met several well-known problems:

- Rule-based expert systems depend on a reliable causal model in domains. The formulation of rules is not a straignhtforward matter in certain domains where the completed comprehensible theory does not exist. Knowledge elicitation is often referred to as the greatest bottleneck of rule-based expert systems. It may have difficulty generating descision rules.

- The rules may be useful for the most common situations we encountered, but we cannot help facing many situations that violate the bounds of the generalizations we made. Since the rules work in an abstract way with missing the detailed information of situations, the systems are unable to look back and know what exactly happened at that moment. Sometimes, rule-based expert systems fail in reaching goals.

- Likewise, once new problems are outside of the rule base, because of the lack of detailed information, rule-based expert systems are unable to make a comparison between current problem and previous experiences to generate or update the rules, so that the rule-based expert systems hardly perform learning.

- Building a rule-based expert system is a difficult process requiring special skills and good understanding in the domain. It often takes a long time to implement.

In order to overcome the limitations shown by rule-based expert systems, over the last few years, using CBR to deal with an open problem has increasingly attracted attention. CBR takes a very different view from other AI approaches. In CBR, the knowledge base is not formed by generalized rules but a collection of stored cases (case base), recording the specific situation of domains. The solutions are generated not by chaining, but by retrieving the most similar cases from case base and adapting them to fit new situations. Thus some advantages with CBR in dealing with weak theory domain are shown as below:

- CBR systems do not require an explicit domain model. CBR is able to save the specific knowledge of previous experiences in its case base. This avoids the bottleneck of decomposing domain knowledge into a

generalized form. A new problem is solved through comparing experience to experience, finding the most similar ones and reusing them.

- CBR is also an incremental learning process. After each time a problem has been solved, learning can be carried out through saving it to the case base, so that the new case will immediately be available for future problems.

- CBR systems can be quickly developed; since cases capture much of the knowledge implicitly, a deep understanding of the domain is not essential for developing CBR applications.

In conclusion, rule-based expert systems are suitable in narrow, well-understood and stable domains, whereas CBR systems might be a better choice on where a domain problem is poorly understood and dynamic. For this thesis, due to the characters and complexity of AHU, we proposed CBR schemes to represent the domain knowledge of AHU. A more detailed discussion of CBR and the domain problem will be given in later chapters.

Chapter 3.

Key Issues in CBR processing

3.1. Overview

The aim of this chapter is to provide a comprehensive overview of CBR techniques and to discuss the essential issues that we should be concerned with in the development of CBR applications. CBR is a knowledge reasoning system that in many respects is fundamentally different from rule-based systems, instead of relying solely on general knowledge of a domain problem, or making association along generalized relationships between problem descriptions and conclusions, CBR is able to utilize the specific knowledge of previous experiences [12].

It has been widely accepted that CBR is not only a powerful method for computer reasoning, but also a common behavior pattern for humans to solve problems in their daily life. This chapter discusses several topics: Section 3.2 gives several definitions of CBR. Section 3.3 briefly introduces the history of CBR. In section 3.4, the CBR cycle is introduced. The CBR cycle can be seen as a fundamental principle to develop CBR applications. Section 3.5 states some major sub-fields

of CBR, such as case representation, similarity measure, and learning mechanisms.

3.2. Definition of Case-Based Reasoning

There are some widely accepted definitions of CBR. Several ones are given below:

- Case-based reasoning solves a new problem by remembering a previous similar situation and by reusing information and knowledge of that situation. [13]

- Case-based reasoning is reasoning by remembering. [14]

- Case-based reasoning solves new problems by adapting solutions that were used to solve old problems. [15]

- Case-based reasoning is an approach to problem solving and learning. [16]

As the above definitions indicate, CBR is based on the intuition that a new problem is often similar to previously encountered problems, and therefore, that past solutions may be reused in the current situation. As a matter of fact, this is also a powerful and frequent way for humans to reuse or modify past experience in problem solving. Everyone has vast experience in working out a problem during daily life in terms of past experiences. More specifically, while people are facing a new problem, the first thing we usually do is to scan our memory and check out if we have encountered a similar situation before. For instance, if you want to make a copy of your textbook at school, before you go to the copy room, it would not be necessary to explicitly plan how to get to there; you just take the route that you usually go (find a solution from experiences). If there are a lot of people standing in line for service, you may remind yourself of how you avoided

a long waiting time under the same circumstances in the past, such as finding another copy machine on a different floor. If it were a success, probably you would go again (once again, find a solution from experiences). The key assumption is that if two problems look alike, then the solutions to these problems is frequently close as well.

3.3. A Short History for CBR

The philosophical roots of CBR could perhaps be many, but what is not in doubt is that the research of Roger Schank and his group at Yale University is known to be the origin of CBR. In 1977, Schank wrote scripts for knowledge representation. The scripts were proposed as a structure for conceptual memory, describing information about stereotypical events. However, experiments on scripts showed that they were not a complete theory of memory representation. In 1982, Schank developed the Dynamic Memory Theory and Memory Organization Pack Theory (MOP) [15]. In 1983, Janet Kolodner, who is a member of Schank's group, developed the first CBR system called CYRUS [16], which was based on Schank's dynamic memory and MOP theory. It was a question-answering system with knowledge of various travels and meetings of former US Secretary of State, Cyrus Vance. Between 1985 and 1992, there appeared other systems that were based on Schank's theories, including MEDIATOR [17], PERSUADER [18], CHEF [19], CASEY [20] and JULIA [21].

Between 1986 and 1989, an early alternative approach came from Bruce Porter and his group at the University of Texas at Austin. This work initially addressed the machine-learning problem of concept learning for classification tasks. This led to the development of the PROTOS system [22], which emphasized integrating general domain knowledge and specific case knowledge into a single case-memory model. Between 1988 and 1992, another significant contribution to the CBR field was the work done by Edwina Rissland and her group at the university of Massachusetts. They were primarily interested in reasoning for law application

cases. Cases are not used here to produce a single answer, but to interpret a situation in practice, and to produce and assess arguments for both sides in court. Their implementation system was HYPO, introduced in [23].

In the early 1990s, the US Defense Advanced Research Projects Agency program (DARPA) funded a series of workshops on CBR and supported the development of a CBR tool called ReMind [24]. This tool marked the transition of CBR from purely academic research in cognitive science and artificial intelligence into the commercial arena. After ReMind joined in the marketplace, several other tools were almost immediately released, including Kate [25], CBR-Works [26], CBR-tools [27], etc.

In Europe, research on CBR was taken up a little later than in the US. The CBR work seems to have been strongly coupled to expert systems development and knowledge acquisition research than in the US. Between 1988 and 1992, Michael M. Richter and his group developed the PATDEX system [28] in Europe at the University Kaiserslautern in Germany. The main purpose of this system was to use CBR to do technical diagnosis. In 1990, Ramon Mantaras and Enric Plaza developed a case-based reasoning system for medical diagnosis [29]. In 1991, Angar Aamodt developed the system CREEK [30], at the University Trondheim in Norway. This system emphasized the integration of cases and general knowledge.

The history of CBR is not a long one compared to other knowledge-based systems such as rule-based expert system or logical programming. Since the theory of CBR is intuitive nature and the relevant implementation of CBR applications is simple, a number of CBR applications have been built in a wide range of domains during nearly a decade, including financial analysis, risk assessment, technical maintenance, quality control, medical diagnosis, and software support systems.

3.4. CBR Cycle

The most popular process model for CBR is shown in Figure 3-1. It is called CBR cycle [13]. As the highest level of generality, the CBR cycle provides the fundamental guideline of CBR implementations. As shown in Figure, the CBR cycle consists of four major sequential processes, which are the retrieval process, the reuse process, the revise process and the retain process:

- **Retrieve** the most similar case to the new problem.

- **Reuse** (adapt) the information and knowledge in that case to solve the new problem. The selected best case has to be adapted when it does not perfectly match the new problem.

- **Revise** (evaluate) the proposed solution. A CBR usually requires some feedback by asking a human expert to know what is right and what is wrong.

- **Retain** (learn) the part of this experience that it is likely to be saved into a case base for future problem solving. CBR can learn from either successful solutions or failed ones.

In Figure 3-1, the event that triggers the whole process of CBR is the appearance of a new problem; the first step is to retrieve a case that is the most similar to the new problem from the case base. For this step, it is necessary to have an effective procedure that explores the case base and assigns similarity value to each case. As we assume that similar problems have similar solutions, we can simply reuse the solution enclosed in retrieved cases. Although the retrieved case is the most similar one to the new problem, it might not be identical. If necessary, adaptation could be done in the reuse step by slightly adjusting the past solution to account for differences between the two problems. In reality, it is not always so easy to get a reliable solution. Therefore, it's very important to evaluate the suggested

solution from the revise step. After evaluating the result, if the adapted solution works, the last step is to retain the valuable confirmed solution and the problem just solved as a new case. If the solution fails, in this situation, the new case is formed with the problem just solved and the repaired solution, so that the same failure can be avoided next time. The new case will be available for retrieval in later problem solving. As indicated in the Figure, general knowledge plays a part of role in the case base to support the CBR processes. General knowledge usually means general domain-independent knowledge, as opposed to specific knowledge embodied in each case. This support may range from very weak (or none) to very strong depending on the type of CBR applications.

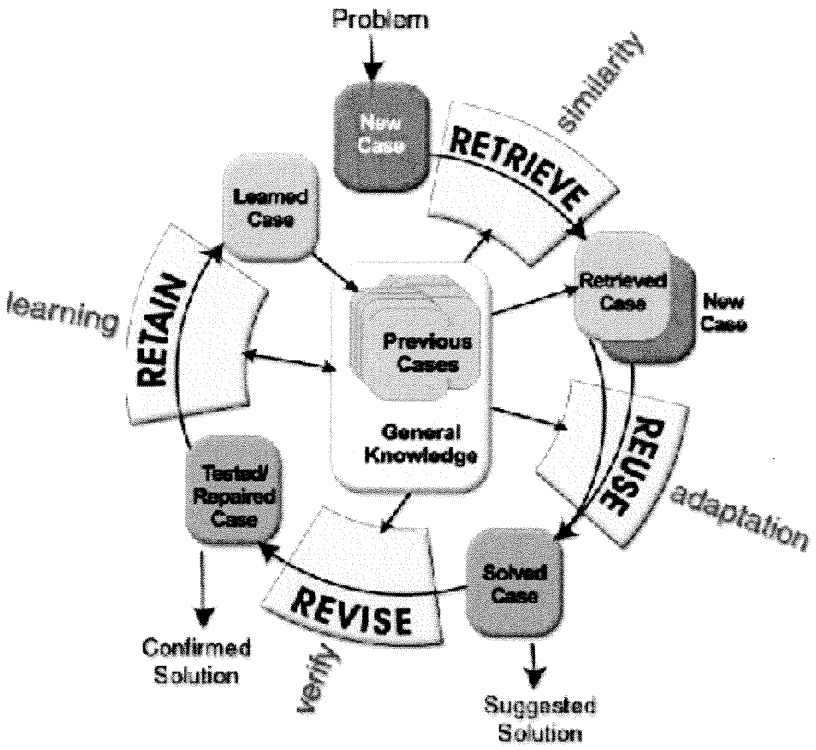


Figure 3 - 1 CBR cycle

As a matter of fact, not all CBR applications use all of the above steps. In some, there is no adaptation step; the retrieved solution is already known to be good

enough without modification. In others, there is no case retain step; the case base is mature and provides adequate coverage for problems in the domain.

In general, CBR applications are often divided into two classes: classification or decision suggestion. The applications of CBR classification use prior cases as reference for classifying new problems. For classification tasks, all necessary information has to be available when the CBR process starts. The solutions of CBR classification are clearly defined. The CBR processes map the new problem to a set of given cases, which can be looked at as static goal definition. Examples are risk assessment, technical and medical diagnosis. The applications of CBR decision suggestion use prior cases to suggest solutions that might apply to new circumstance. The applications of CBR decision support are different from the classifications by the necessity of representing more general knowledge and the allowance of users' interaction to a very high degree. Normally, users are required to answer a list of questions that helps narrow the number of cases. The more accurate solution is generated from dynamic goal definition. Examples are looking for the right financial investment, or travel advisor.

3.5. Major Sub-fields in CBR

On the basis of the CBR process model described above, it is now rather easy to have clues as to what are the key issues in building a usable CBR application. First of all, that may be to define how to represent the cases to capture the true meaning of the domain. Next, in order to retrieve cases efficiently, it is import to develop an indexing method to bring back a certain number of similar cases in a rather short time. Then, it would be to design a robust function to measure the similarity between the new problem and cases. If the domain is of a kind where it makes sense to adapt a suggested solution to the actual situation, then another key issue is the process of adaptation. Later, if necessary, the last issue might refer to the ability of CBR systems to carry out learning.

3.5.1. Case Representation

Technically, a case is the place where it stores previous experienced situations, usually including problem descriptions, solutions and outcomes.

- **Problem descriptions:** The state of the world while the case was happening and what problem needed to be solved at the time.

- **Solutions:** The state of derived solution to the problem.

- **Outcomes:** The state of the world after the case occurred.

Another way to describe case representation is to visualize the structure in terms of the problem space and the solution space [31]. Figure 3-2 illustrates the structure space. According to this structure, the description of problems resides in the problem space. The retrieval process identifies a set of relevant features between the descriptions of a new problem and solved problems. The CBR systems use similarity functions to find the closest matching one. As soon as the matching case is found on the problem space, a link contained in that case can immediately reflect its solution on the solution space. The solution may be adapted or directly used to solve the new problem.

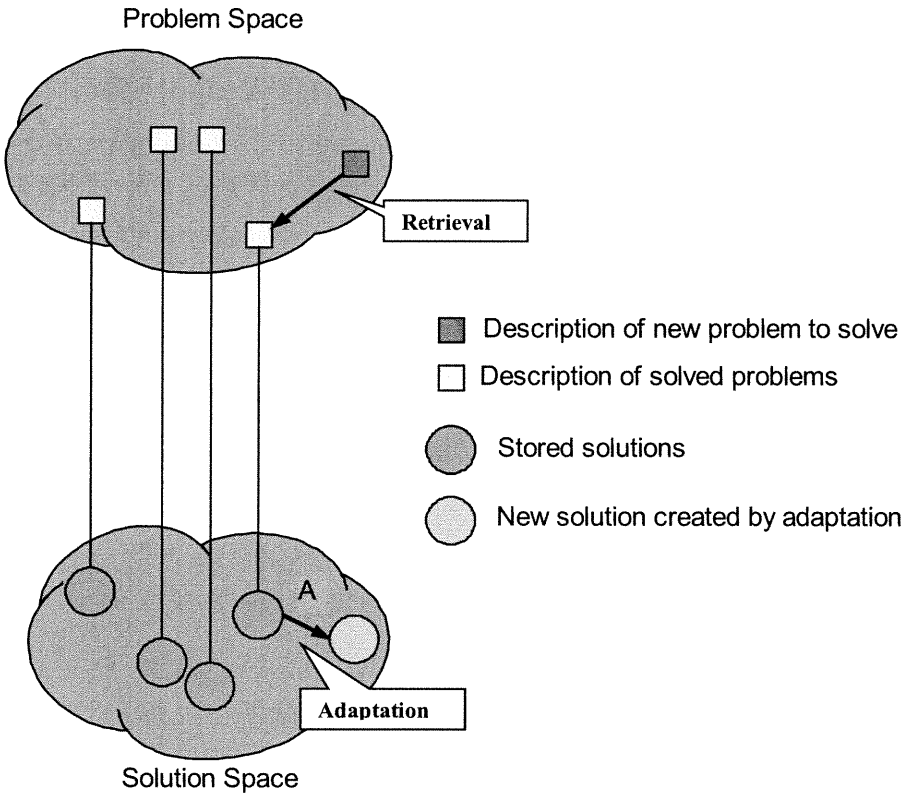


Figure 3 - 2 Problem and solution spaces

3.5.2. Case Retrieval

Case retrieval is a process that a retrieval algorithm retrieves the most similar cases to the current problem. The case retrieval requires a combination of search and matching. In general, two techniques are currently used by commercial CBR applications: nearest neighbour retrieval and induction retrieval.

- **Nearest neighbor retrieval** is the most popular method to measure similarity. Nearest neighbor retrieval is used to calculate the similarity between a new problem and past cases in the case base. Both the new problem and the cases consist of a list of attributes or objects with values. The algorithms of nearest neighbor retrieval determine the similarity between the new problem and the cases in terms of these values. The

nearest neighbor algorithms all work in a similar fashion starting from the local similarity measure. The local similarity is the computation similarity upon same attributes between the new problem and the past cases. The calculations depend on the type of attributes. Similarity can be calculated for either numeric values or non-numeric values. The following are common methods to calculate local similarity:

Local Similarity

Numeric $$sim(a,b) = \frac{|a-b|}{range}$$

Symbolic $$sim(a,b) = \begin{cases} 1 & if \quad a = b \\ 0 & if \quad a \neq b \end{cases}$$

Where *range* is the absolute value of difference between the upper and lower boundary of the set.

Once a set of local similarities has been calculated for each attribute, then the nearest neighbor algorithms go to count the global similarity. A typical equation is used to compute the global similarity [32] as below:

Global Similarity:

Weight Block-City:

$$similarity(A,B) = \frac{\sum_{i=1}^{n} w_i \times sim(a_i, b_i)}{\sum_{i=1}^{n} w_i}$$

Where *sim* is the local similarity function, and a_i and b_i are the values for attribute i in the new problem and the selected case respectively. This measure may be multiplied by a weighting factor w_i, which indicates the importance of the value of this attribute in the case. Then the sum of the similarity of all attributes is calculated to provide a measure of the similarity of that case in the case base. This calculation is repeated for every case in the case base to rank a similarity value to the new problem. The final result of nearest neighbor algorithms is usually normalized to fall within a range of zero to one where zero is totally dissimilar and one is an exact match.

Figure 3-3 displays a simple example for nearest neighbor algorithms. In this two dimensional space, *case3* is selected as the nearest neighbor because *similarity* (*NP, case3*) > *similarity* (*NP, case1*) and *similarity* (*NP, case3*) > *similarity* (*NP, case2*).

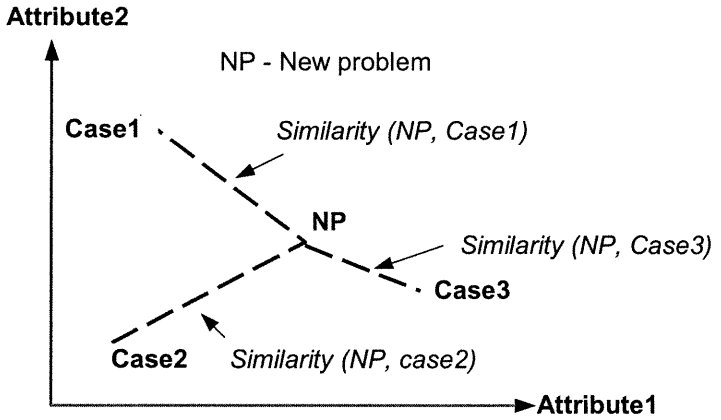


Figure 3 - 3 Nearest neighbor algorithms

- **Induction retrieval** is a technology that utilizes a collection of sample cases as patterns to solve new problems. The sample cases, often referred to actual historical data or synthetic cases, can be organized into a decision

tree type structure. The induction algorithms identify patterns amongst the sample cases and classify a new problem into clusters. Each cluster contains the cases that have at least one attribute similarly. A requirement of induction is that the attributes of the sample cases are well defined. This approach is very useful when a single attribute in cases is required as a solution, and that attribute is dependent upon others. Here is a completed decision tree (see Figure 3-4) generated from the data in Table 3-1. The task is to predict the health status of people according to three attributes, including ages, physical activities and smoking.

Case No.	Health Status	Ages	Physical Activity (Average hours per week)	Smoking (Cigarettes per day)
Case 1	Good	34	7	5
Case 2	Very Bad	44	2	25
Case 3	Bad	25	20	0
Case 4	Very Good	36	4	8

Table 3 - 1 Sample Cases

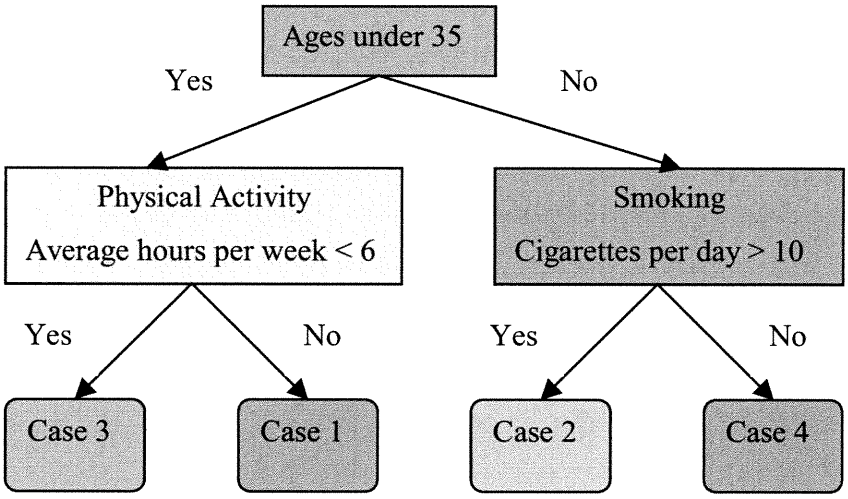


Figure 3 - 4 Decision tree

If a new problem were presented as shown in Table 3-2, to determine the health status of this case, the algorithm would traverse the decision tree and search for the best matching case in the case base. For the given age, the algorithm first chooses the left branch. After this, the algorithm traverses to the node of physical activity and selects the right branch according the average hours per week. The algorithm can therefore predict that the best matching case is the Case No.1, in which the case suggests that the health status of this person is good because the status of Case No.1 is good.

Case No.	Health Status	Ages	Physical Activity (Average hours per week)	Smoking (Cigarettes per day)
Problem	?	28	10	10

Table 3 - 2 A new problem

The induction retrievals are useful where the retrieved goal or the case pattern is well defined and there are enough of the sample cases of each type of goals with which to perform inductive comparisons.

Nearest neighbor retrieval and induction retrieval are widely applied in CBR applications and tools. The choice between two of them in CBR applications requires experience and experimentation. Usually, it is a good choice using nearest neighbor retrievals without any pre-indexing [31]. If retrieval time becomes an important issue, induction retrieval is preferable, because the retrieval time using inductive index trees is extremely quick and only increases slowly as the number of cases in the case base increases. However, induction retrieval has one major disadvantage: if data in the new problem is missing or unknown, it may be impossible to retrieve a case at all. In some CBR tools, both techniques are used: inductive indexing is used to retrieve a set of matching cases, then the nearest neighbor algorithm is used to rank the similarity of cases in the set to fit the new problem.

3.5.3. Case Indexing

Any system that is performance based on the selective use of items from a large amount of data must find some way to organize that data so that the right items can be found at the right time. In CBR, this problem of how to ensure effective selective retrieval goes by the name of the index problem [38].

The retrieval of cases is closely related and depends on the indexing method used. The simplest way of doing case retrieval is by sequential search. Clearly, this cannot be recommended for larger amounts of cases. Therefore, more complex problems need one or more index structures to be built, and cases are then searched through traversing the respective index structure. Case indexing involves assigning indexes to cases to facilitate their retrieval. The CBR community proposed several guidelines on indexing [33]:

- Indexes should be predictive, that is, they should address the purposes the case will be used for.

- Indexes should be abstract enough to allow for widening the future use.

- Indexes should be concrete enough to be recognized in future.

3.5.4. Adaptation

After a CBR system retrieves the most similar case from case base, it might perform adaptation on the retrieved case. There are several adaptation strategies that can be used in CBR systems. Some of them are overviewed here.

- Null adaptation: a direct simple technique that applies whatever solution is retrieved to the current problem without adapting it. Null adaptation is useful for problems involving complex reasoning but with a simple solution.

- Simple substitution: a structure adaptation technique that compares specified parameters of the retrieved case and the current problem to modify the solution in an appropriate direction.

- Constraint satisfaction: this method requires a set of constraints to be stored in cases. The adaptation process is accomplished by checking the constraints back and forth to decide whether or not the solution of the case satisfies the constraints of the new problem.

Clearly, the above strategies are very helpful in designing adaptation process. However, so far, generic adaptation methods or algorithms have never yet been exposed. The practical methods of adaptation are still domain-dependent.

3.5.5. Case Evaluation and Learning

Like case adaptation, evaluation of the goodness of retrieved cases may become a problem for CBR applications, because evaluating candidate solutions may require a considerable general domain knowledge and reasoning effort. Although the methods of case evaluation have been developed to judge the quality of solutions in some types of case base, providing the right general evaluation methods is still difficult [14].

A commonly used definition of machine learning is: A learning machine is one that is able to modify itself in such a way as to improve its future performance, either by increasing the efficiency or accuracy by which it performs one of its current tasks, or by allowing the system to perform some new task that it was unable to perform previously [9].

A CBR learns after case evaluation and possible repair by retaining relevant information from a problem just solved. The learning of CBR can be preformed based on either the success or failure of the proposed solution. A CBR learning process can be a combination of adding a new case to the case base, modifying an existing case in the case base, or removing an existing case from the case base.

CBR as a learning paradigm has several technical advantages. One advantage has already been stated, since newly solved problems can be stored in the case base as new cases for later use; then, in the future, when a problem has similar characteristics, the solution to this problem will be remembered but not recomputed. Other advantages are that CBR applications are able to become more competent over time - the same mistakes can be avoided.

Chapter 4.

Fault Detection and Diagnosis in AHU

4.1. Overview

This chapter addresses the domain problem of AHU and strategies for performing the diagnostic task. In section 4.2, the motivation of fault detection and diagnosis is introduced. In the following section 4.3, the AHU system description is given. The functionality of each single element in AHU is described. Section 4.4 states two possible approaches to fault diagnosis. Section 4.5 explains the fault detection and diagnosis scheme and illustrates some typical failures in AHU, so that they can be detected according to the scheme that we proposed. Section 4.6 introduces the architecture of the FDD tool that we developed. Actually, the fault detection and diagnosis in AHU is combined with two tasks, which are fault detections and fault localizations, respectively. This thesis concentrates on fault diagnosis in AHU by using CBR methodology. The reason for choosing to use CBR techniques for this application is that the domain knowledge of AHU is extremely incomplete, and it is almost impossible to generalize a complete causal model. In

section 4.7, the benefits of using the FDD tool to solve detection and diagnosis problems in AHU are discussed.

4.2. Motivation of Fault Detection and Diagnosis

As a result of economic and population growths, energy use in modern buildings has increased rapidly in recent decades. Energy saving has become an important issue since the Global Energy Crisis. HVAC systems are responsible for a significant part of total energy consumption in the world. A HVAC system is typically composed of an AHU and several VAV boxes. As the central equipment of the HVAC system, AHU systems have a major impact on the total energy consumption in the HVAC and the assurance of comfortable conditions in the building space. Due to the complexity of AHU and the long period of running time, faults occurring in AHU systems are common, such as sensor drift, stuck damper or fan failure. All kind of failures can lead to excessive energy consumption, discomforts to human occupants, and increasing premature wear on control equipment.

Over the past decade, due to the reduction in cost and greatly increased capability of computers, the applications of a building energy management system (BEMS) to control AHU systems based on computer auto-control technologies have been widely adopted. AHU systems can be equipped with numerical sensors and computerized control units. The measured signals of all elements in an AHU are currently available in BEMS, which seems to have sufficient capability for monitoring system behaviors. Unfortunately, since these is little effort to isolate the monitoring information into direct and clear data analysis tools, building operators are not always able to notice that the system is running in unexpected manner. Such unexpected performance could go unnoticed for long periods of time. Damages may have been done if one waits until building occupants complain about the lack of comfort. Therefore, there is a high demand for developing a fault detection and diagnostic tool (FDD) to assist building operators

in maintaining the optimal performance of AHU systems. The FDD is used to continuously monitor the performance of an AHU in terms of all the information input from BEMS.

4.3. AHU System Description

The principal task of AHU is to deliver an appropriate air condition requested by human occupants in the building space. This task has to be accomplished in a reliable and economical way. Normally, the AHU systems consist of outdoor, mixing and exhaust air dampers, cooling and heating coils, an air filter section, supply and return fans, and a humidifier. A simplified schematic diagram of an AHU is shown in Figure 4-1. Outside fresh air from a supply air damper suck up to the building, pass through cooling, heating and humidity controllers so that the air can be moisturized, cooled down or heated up, then a supply air fan blows the air into the building space or rooms. At the same time, the air is also pulled back by a return fan from the building space and discharged to the outdoors from an exhaust air damper. A small amount of return air mixes with fresh air in a mixing box to re-enter the building space through a mixing air damper.

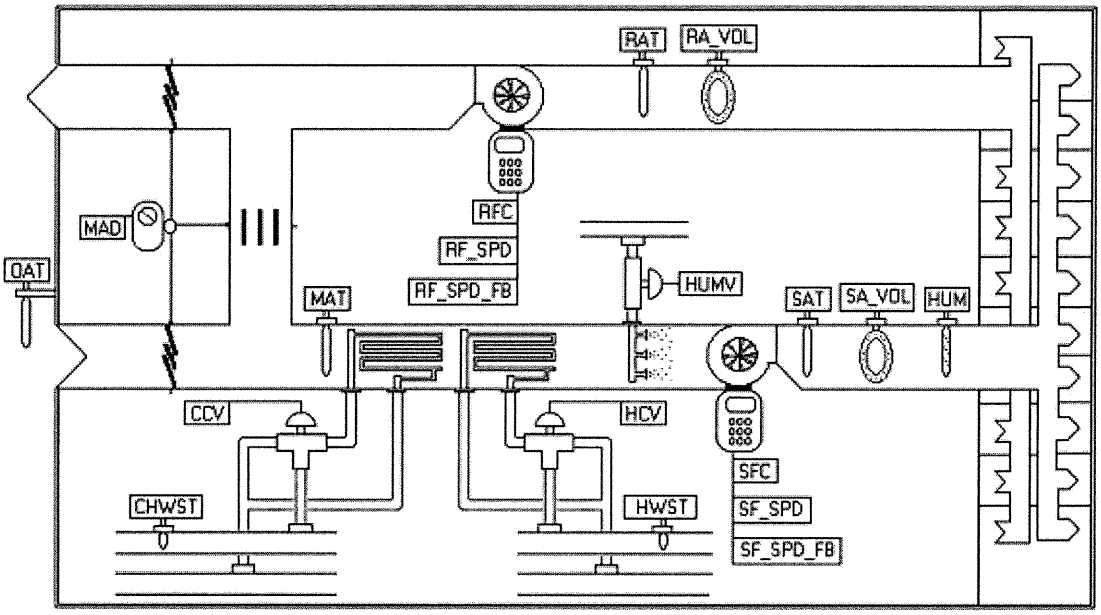


Figure 4 - 1 AHU schematic diagram

In AHU, all of the controllers and sensors can be grouped into four major control loops, including air pressure, airflow, temperature, and humidity. Each control loop is adjusted by a direct digital control system (DDC) with proportional integral derivative (PID) algorithms. The static pressure in the supply duct is controlled to maintain a constant static pressure at each room inlet by air volume sensors, supply and return fans. The desired return airflow (calculated by using supply airflow minus both the airflow through the return fan and the amount of airflow required for building pressurization) is compared the actual return airflow and the difference is controlled by the return fan with a variable speed. The supply air temperature is controlled to maintain an appropriate temperature by cooling or heating coils. An enthalpy control economizer allows cooling with cooler air, and humidification is provided to maintain proper humidity in the return air during the winter [34]. Depending on the outside air temperature, three standard working operation modes in an AHU can be chosen.

- **Heating operation:** Once the cold season is coming, hot water is supplied to the heating coil in the AHU. This coil heats up the supply air so that warm air can be sent to all the building space.

- **Cooling operation:** Once the hot season is in full swing, chilled water is supplied to the cooling coil. This coil cools down the supply air so that cool air can be sent to all the building space.

- **Free cooling operation:** During certain times of year, the cooling required by the building space can be satisfied by the outside air temperature directly. In this case, both the outside and exhaust air dampers are fully open. That is an economical way to provide acceptable air conditions to the building space.

4.4. Approaches to Fault Diagnosis

The approaches to fault diagnosis can roughly be classified into two categories: model-based and knowledge-based diagnosis systems.

Early efforts at developing diagnostic tools have primarily focused on model-based systems. The concept of model-based diagnosis is based on analytical redundancy [35]. Model-based systems use the information from mathematical models in terms of the physical laws or empirical data of the system to generate a shadow system or simulation system to an actual system. The type of models generally may be first principle or black box. For both models, the generic information about the simulation process is embodied in the equations. In the first principle models, the parameters that appear in the equations are used to represent one physical item in the system. By contrast, the equations in the black box are indented to be general enough to model. The simulation system, based on a computational model, is built to run in parallel to an actual system.

Model-based diagnosis systems characterize failures as deviations from the actual system and the computational model simulation system. Both systems during execution have the same input. The output from model-based systems should be the same as that of the actual system (if not, the model has to be modified). The difference between the simulation model and the actual system is called residual. The process to create the residuals is called residuals generation, shown in Figure 4-2. The residuals are used to detect the fault if one occurs in the system. Theoretically, when the system runs under normal conditions, the residual will be zero, and any fault occurring in the system will make the residual non-zero.

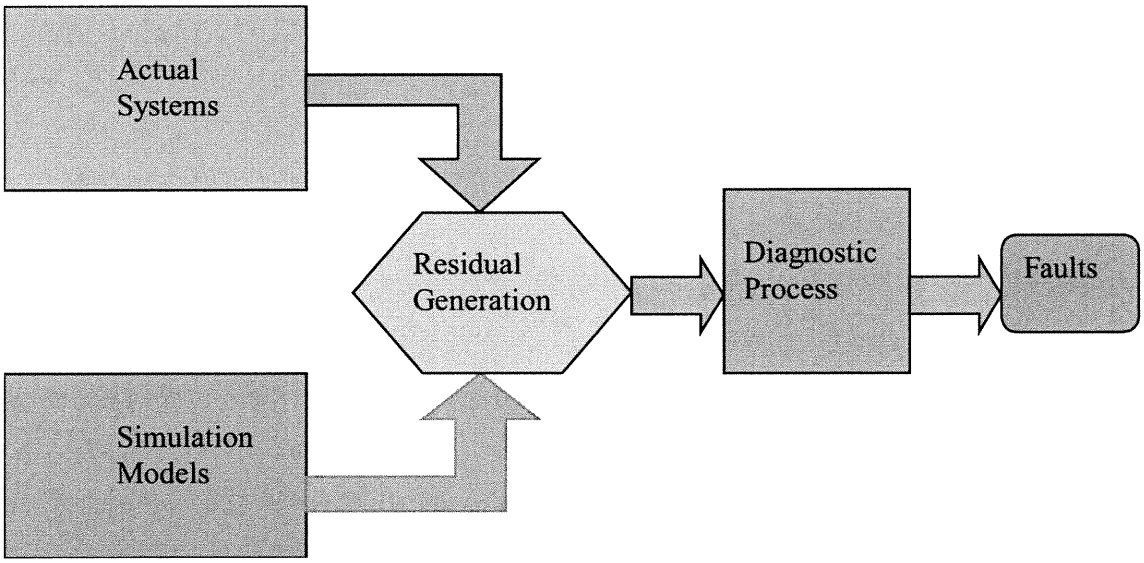


Figure 4 - 2 Model-based diagnostic systems

During the utilization in practice, several shortcomings of model-based diagnosis systems have been revealed. First, notice that the competence of model-based diagnosis systems crucially depends on the model to generate the prediction of the actual system. However, it is very difficult to obtain an adequate model to represent the real and complex systems. Secondly, the computational model in model-based systems usually relies on the control laws used in the actual system. If any changes happened in the control strategy of the actual system, the relating

model would become inadequate or even useless. Thirdly, most model-based systems require a long time to carry out tuning in order to identify the simulation model. These factors all add cost to model-based diagnostic systems that limits their practical applications.

By contrast, knowledge-based systems, including rule-based expert system and CBR, often result in an easier way to construct and have enough flexibility to deal with incremental requirements. Fault detection and diagnosis scheme determining normal or abnormal situations in systems can be defined in a way as being independent of control laws.

4.5. Fault Detection and Diagnosis Scheme

By means of FDD, Fault detection is the determination that the operation of the building is incorrect or unacceptable in some respects, whereas, fault diagnosis is the identification or localization of the cause of failure. To carry out FDD relies on how to define the abnormal or normal behaviors of AHU, because raw information directly given by the BEMS system does not suffice to know whether the system can run under an expected situation or not. The FDD scheme is a sort of reference to indicate whether the actual performance is satisfied. In this thesis, we use the calculation of the difference between the actual measured sensor values and variable control signals, or certain set points, as the FDD scheme; namely, residuals or innovations (we should not get confused with those residuals which are different from the ones that we explained in model-based systems. The data used to calculate the residuals here all come from the AHU itself, not somewhere else). The residuals can be looked at as signatures or symptoms to identify a particular fault. The process of fault detection is used to continuously compare the deviation of the residuals to predetermined thresholds. Faults are detected once a specified threshold limit is exceeded. Faults diagnosis is a task of finding the causes of symptoms or malfunctions. The diagnostic procedure can

also use a residuals list of behavior characteristics to infer the cause of malfunctions.

4.5.1. Residual Description

The data inputting to the FDD tool are all from the BEMS for controls and energy monitoring. The input data is divided into three parts. The first part is certain set points, such as supply air temperature set point and return air humidity set point, which are entered by building operators to indicate an expected air condition. The second part is the measured sensors data from the AHU, such as supply air temperature, mixing air temperature, return air temperature and return air humidity, which reflect the actual running situation of the AHU. The third part is the control signals and feedback generated by variable control units in the AHU, such as the status of the air dampers, fan speeds, steam and chilled water valves and positions.

Basically, the residuals are defined as the difference between actual measured values and control signals, or associated set points. There are ten residuals existing in current AHU systems.

- **Air Temperature**

  The residual of the supply air temperature, R_{SAT} is defined as

  $$R_{SAT} = T_{SA} - SP_{SA};$$

  Where T_{SA} is the sensor value of supply air temperature and SP_{SA} is the supply air temperature set point.

- **Temperature Controllers**

  The residual for the heating coil valve R_{HCV} is defined as the difference between the control signal C_{HCV} and the measured status of the valve S_{HCV}.

  $$R_{HCV} = C_{HCV} - S_{HCV};$$

The residual for the cooling coil valve R_{CCV} is defined as the difference between the control signal C_{CCV} and the measured status of the valve S_{CCV}.

$$R_{CCV} = C_{CCV} - S_{CCV};$$

- **Damper**

The residual of the outside air damper R_{OAD} is defined as the difference between the control signal C_{OAD} and the measured position of the damper P_{OAD}.

$$R_{OAD} = C_{OAD} - P_{OAD};$$

The residual of the return air damper R_{RAD} is defined as the difference between the control signal C_{RAD} and the measured position of the damper P_{RAD}.

$$R_{RAD} = C_{RAD} - P_{RAD};$$

The residual of the mixing air damper R_{MAD} is defined as the difference between the control signal C_{MAD} and the measured position of the damper P_{MAD}.

$$R_{MAD} = C_{MAD} - P_{MAD};$$

- **FAN**

The residual of the supply air fan R_{SAF} is defined as the difference between the control signal C_{SAF} and the measured value of the fan speed S_{SAF}.

$$R_{SAF} = C_{SAF} - S_{SAF};$$

The residual of the return air fan R_{RAF} is defined as the difference between the control signal C_{RAF} and the measured value of the fan speed S_{RAF}.

$$R_{RAF} = C_{RAF} - S_{RAF};$$

- **Humidity**

The residual of the return air humidity, R_{RAH} is defined as

$$R_{RAH} = H_{RA} - SP_{RA};$$

Where H_{RA} is the return air humidity and SP_{RA} is the return air humidity set point.

The residual for the humidifier control R_{HC} is defined as the difference between the control signal C_{HC} and the measured status of the valve S_{HC}.

$$R_{HC} = C_{HC} - S_{HC};$$

The process of fault detection is used to continuously perform threshold checking by calculating the deviation of the residuals to thresholds. The thresholds can usually be determined from pre-defined values. The concept of threshold is very straightforward. If a residual is greater than an upper limit threshold limit, or is lower than a lower threshold limit, then the process shows that the AHU system is out of the normal state and a fault is presumed to have occurred.

4.5.2. Faults Description

How do the residuals work? In this section, some faults denoting typical failures in an AHU system are described. The dominant residuals of each fault are also described.

- **Example one:** There is a failure of the supply fan. During normal operation both the supply fan and the return fan are controlled to maintain a static pressure in the air duct. The fault causes the supply fan rotational speed to decrease so that the supply air pressure decreases, and the control signal to the supply fan raises automatically in an attempt to offset the decreasing supply air pressure. At the same time, the control signal for the return fan decreases in order to maintain the airflow between the supply and return air ducts; however, this condition cannot be achieved due to the failure of the supply fan so that the dominant residual of R_{SAF} and R_{RAF}

will finally exceed the limit of the threshold. In addition, because there is no airflow, the supply air temperature also goes up gradually, as a result, the temperature control coil R_{HCV} or R_{CCV} (heating or cooling, depends on what mode is on) and the supply air temperature set point R_{SAT} might be outside the normal range as well.

- **Example two:** There is a failure of the return fan. The fault causes the return fan rotational speed to decrease to zero, so that difference of airflow between the supply and return ducts increases, and the control signal to the return fan raises to its maximum value in an attempt to offset the increasing flow difference. However, this condition cannot be achieved due to the failure of the return fan. Thus, the dominant residual for this fault is R_{RAF}.

- **Example three:** The stuck of heating or cooling coil valve causes supply air temperature to decrease or increase gradually. In an attempt to offset the difference between the supply air temperature and associated set point, a control signal is sent to the coil valve to compensate. However, as the valve is stuck, the position does not change. Over time, R_{HCV} or R_{CCV}, and R_{SAT} could be outside the normal value for this fault.

Currently, there are at least thirty faults declared in AHU. It is no surprise that some of them might have the same list of dominant residuals. Using this FDD scheme makes it possible to reveal all potential faults, but it is obviously not good enough to isolate the cause of failure, as this scheme is not able to identify the fault unambiguously; extra knowledge of the AHU is needed.

4.6. Architecture of the FDD Tool

Rule-based expert systems are an efficient way to perform deduction; as rule-base is well designed, it is able to make a decision quickly; CBR systems are often

suitable for where expert systems find it hard to generate the rules from the incomplete theoretical basis of domains. In this project, these two approaches are integrated into a powerful FDD application. We maintain the power of the rule-based expert system and CBR system in reasoning with knowledge, respectively. We believe it can significantly enhance the ability of the FDD tool. The architecture of the FDD tool is shown in Figure 4-3.

For fault detection, it is only necessary to determine whether the performance is incorrect or unsatisfactory. The knowledge of how a particular fault affects performance is not required. Since the existence of the FDD scheme, a rule-based expert system is suitable for the generalization of all the definitions of residuals as decision rules to determine all possible faults in the AHU. In fact, a rule-based expert system based on the FDD scheme has been developed successfully by CEDRL. This rule-based expert system is primarily intended to detect all possible faults as well as provide a low-level diagnosis. In other words, the main goal of this expert system is that it emphasizes the fault detection by using rules deduction and has some basic diagnostic support.

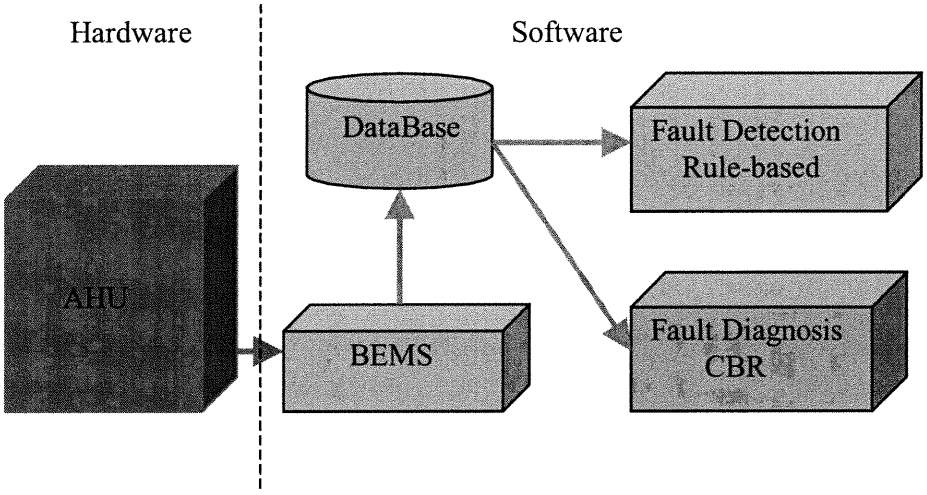


Figure 4 - 3 FDD Architecture

Fault diagnosis is more difficult than fault detection because it requires a more detailed analysis of residuals and associated information to determine which of the possible causes of faulty behavior are consistent with the observed behavior. Due to the dynamic behavior of AHU is non-linear and poorly understood, it does not suffice to explain the interaction among all of the elements in AHU if it only depends on the FDD residuals scheme. The different faults might produce the same symptoms (residuals), and the causal information for why elements fail is incomplete. In this case, using a rule-based expert system as a diagnosis system would be difficult, and is likely to produce incomplete or inaccurate results. Moreover, all elements in AHU take affect in a single duct so that the control behavior can be heavily influenced each other, thus, it has to look over all the detailed information of entire system to determine a fault. As a result, we decided to use CBR techniques to achieve the task of fault diagnosis.

CBR diagnosis systems are able to express specific and concrete diagnostic information as cases. A case is a single instance of the experience of the AHU. The process of retrieval is to search the entire case base and compare each case with a given problem. The comparison between the case and problem not only limits checking to the list of residuals, but also makes possible the calculation of the similarity measure more specific; because of this, no information about the AHU is lost in the cases; all the information of individual experience is still available in the case base. Lastly, there is plenty of time-series data produced by building energy management systems (BEMS) during the monitoring period. All these data can be easily transformed into cases.

4.7. Benefits of the FDD TOOL

The capability to rapidly detect and diagnose faults enables us to optimize the real-time performance of the AHU so that the whole system can operate in the intended manner. Some of benefits are summarized as below:

- The indoor environment is more comfortable and healthy.
- Diminishment of the work complexity of building operators.
- Reduction in energy consumption.
- Longer equipment life
- Maintenance costs reduced

Chapter 5.

CBR Diagnosis System

5.1. Overview

This chapter gradually introduces the CBR approach to the design of a diagnosis system for the AHU. Our approach is based on the expressiveness of unified modeling language (UML), which is the standard of an object-oriented modeling language. We created a set of diagrams in UML notation illustrating the each step of the CBR process that we developed. Once more, the major intention of this thesis is not only to develop a prototype of a CBR diagnosis system for the AHU, but also to construct a generic diagnostic framework for various kinds of domains. The basic idea is the development of a reusable and extensible environment that integrates a domain-independent case representation model with a skeleton of reasoning processes. Section 5.2 explains the prototype of the CBR diagnosis system. Section 5.3 introduces unified modeling language (UML), which is used to achieve all design tasks. Section 5.4 shows the development of the CBR framework.

5.2. Prototype of CBR Diagnosis System

After struggling a long time to investigate the domain, the system requirements and basic functionalities of a diagnosis system finally became clear. Now it is the time to put in place the first brick. In the prototype of the CBR diagnosis system that we proposed, two aspects of the design tasks are addressed: one as a case representation and another as a reasoning process. Before explaining the design tasks in detail, the whole picture for this prototype is illustrated in Figure 5-1.

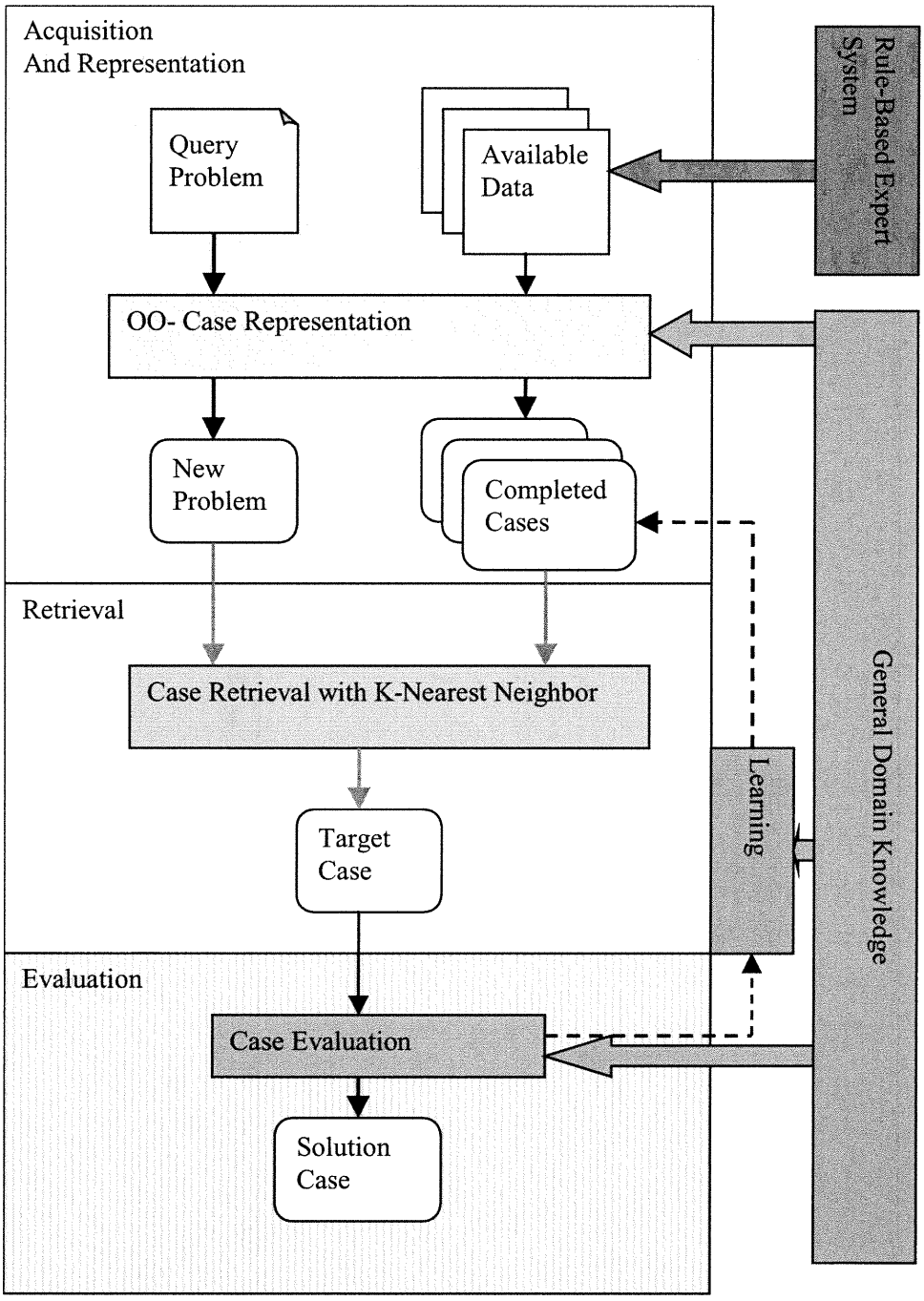


Figure 5 - 1 Prototype of CBR diagnosis system

Case representation is used to capture certain means of the domains. In this prototype, we use the object-oriented concepts to realize domain modeling.

Object-oriented modeling is a promising technique since it is considered to be conceptually mature and expressive enough to represent various types of domain knowledge. As shown in Figure 5-1, going through the case representation layer, a query problem and previous available data are formulated into a unique case representing structure. The available data denotes previous experiences that come from the BEMS and rule-based expert system. CBR systems have a basic problem that they cannot work alone if there are no available experiences, especially, in the initial running period of CBR systems. So before CBR systems are ready for running up, the previous experiences about the AHU has to be refined and translated into a collection of completed cases (case base). A completed case consists of problem descriptions and a known solution. The query problem denotes the current situation of the AHU that comes from the environment, more specifically, from the BEMS. During running time, the query problem also needs to be translated into the new problem that has the same representing structure with the completed case, but without the solution, which we should figure out by comparing the problem descriptions between the new problem and the completed cases.

The reasoning process that we defined in this prototype is fundamentally based on the CBR cycle, even though we are still far from reaching its full coverage. The reasoning process started by the following steps:

- Step1. Obtaining a new problem as input.

- Step2. Retrieving several completed cases that best match the new problem. K-nearest neighbor algorithm is chosen to rank a similarity value to each completed case. The completed case with the highest value is selected as a target case. The fault number enclosed with the target case is proposed as the solution without adaptation.

- Step3. Evaluating the solution by human experts. If the proposed solution is accepted, the reasoning process will stop with success. If the solution is rejected, the process will ask human experts to repair the case and add it to the case base.

In the CBR approach, the process of adaptation by slightly changing the proposed solution to fit better with the new problem is the one of key issues in development of CBR applications. However, the diagnostic solution in this prototype is only about the fault number. The meaning behind this number is explained by the expert system. Because the solution is too simple, any small change is likely to mean that the result by the retrieval phase has to be changed. Consequently, we did not develop the adaptation process since it appears that it is not relevant to the current requirement of the diagnosis system. Moreover, the process of adaptation could be involved if we were asked to go further and to provide the solution of how to fix failures in the AHU.

As it is the first version of the CBR diagnosis system, this prototype needs to involve the human experts who vote for whether the solution is correct or not. The human experts are also asked to modify the target case if the solution is not good enough. To automatically evaluate the solution and to perform learning that concerns how to make good use of the general domain knowledge will be discussed in future works and realized in a later version of the diagnosis system.

5.3. Unified Modeling Language (UML)

By illustration in the last section, the major design tasks can be separated into two aspects: one is to design the case representation that is used to conceptualize and represent the domain problem; another is to design the functionalities of the reasoning process that is used to manipulate the representation of the problem in order to draw a solution. Both of them need to have a good design model, which

is extremely helpful to make better understanding of domain concepts and to simplify program development.

In this thesis, we utilized UML as a modeling language in case representation and the reasoning process design. The UML is a language for specifying, visualizing and constructing the artifacts of a software system [36]. Besides, UML is a general notation system aimed at modeling system using object-oriented concepts. The UML provides several diagrams for almost every sort of situation we need to deal with in software designs. The power of using UML to support the development of CBR applications not only limits the case representation and the reasoning process design, but also spreads to all phases of system implementation. Object-oriented modeling techniques are intimately related with incremental development and reuse. The application based on the object-oriented modeling techniques can be easily update with the times, since the changes in a part of the code of some classes have a minimum effect in the rest of the classes. Likewise, object-oriented modeling techniques have proven to be flexible and efficient in handling complex problems. They are able to support a different level of abstraction to explicitly express domain knowledge in a class hierarchy instead of a simple structure tree, and fit to various different types of domain. Furthermore, UML is an industry standard modeling language supported by the Object Management Group (OMG) and broadly recognized by the software industry. More importantly, there are several well-known tools in terms of UML, such as Rational Rose and Together-J, which usually provide a convenient graphical interface, and a powerful procedure for generating source code, especially in Java as we have chosen it as an implemented language. Using the same implemented language both in the case representation and the reasoning process facilitates the development of a computer program that manipulates the data in the representation.

5.4. CBR Framework

To gain higher reusability during the development of a CBR diagnosis system for an AHU, we are also concerned with building a generic CBR diagnostic system for various kinds of domains. The basic idea is to integrate a domain-independent case representation structure and a skeleton of reasoning engine into a framework. The greatest desire for this framework is to facilitate the development of later CBR applications by reusing previous designs and implementations.

A practical and useful framework basically consists of two types of software entities: classes and methods. Both kinds of entities in object-oriented concepts are not independent but they reflect two different reusability issues: the reuse of abstract data types and the reuse of functions. In our CBR framework approach, we defined two types of classes: one including a set of abstract classes corresponding to a generic case base structure, and the other also including a set of abstract classes corresponding to common and popular reasoning methods. With object-oriented concepts in mind, it is necessary to declare a set of interfaces to make boundaries among those classes. During the execution, the manipulation of an object is only possible through its defined interface. The variables and methods in the classes are hidden. As a result, the implementation can be changed without affecting the existing program code that uses the interface of the classes.

Because the CBR framework is supposed to be reused in various kinds of domains, it should try to keep as general as possible. However, since CBR applications always use specific domain knowledge to solve problems, sometimes there is no evidence of distinguishing domain-specific and domain-independent knowledge in CBR applications. What information should be kept generalized and what should remain customized, and how should we build a map over this gap between the domain-specific and domain-independent knowledge? This is a crucial problem in developing a generic framework for CBR applications.

As mentioned above, the CBR framework is composed of the case representation and the reasoning procedure. The following Figure 5 – 2 is a simplification of the description of our design.

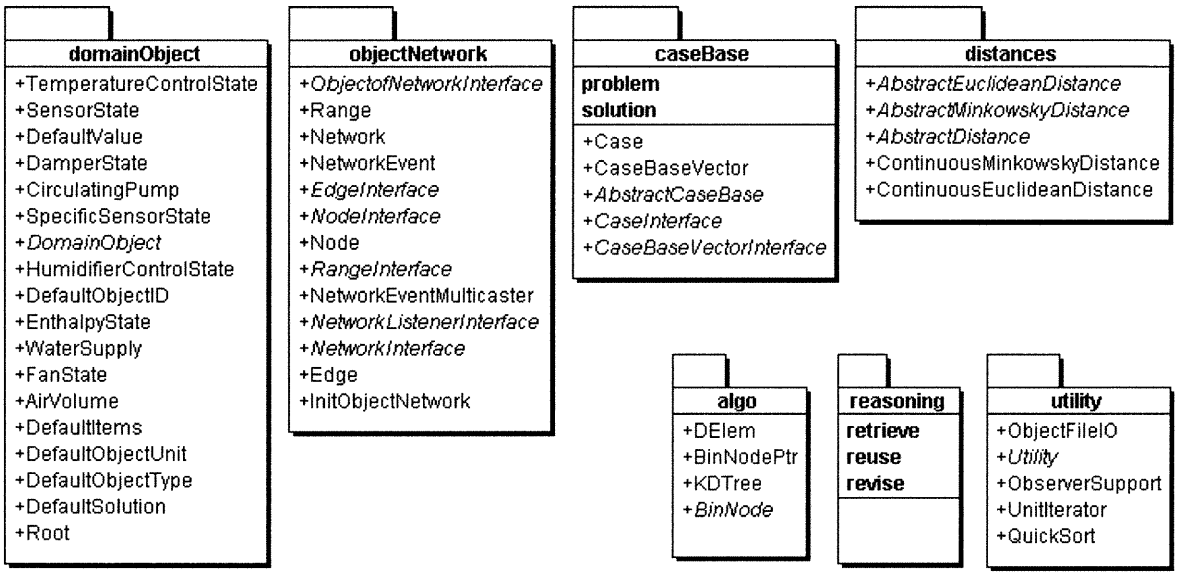


Figure 5 - 2 Main Packages

5.4.1. Case Representation

For the sake of the generalized and specified, we divided case representation into three-layers as shown in Figure 5-3. The top layer in this case representation is the package of the Case-Base, which emphasizes the conceptual structure of the case base rather than the domain specific. A case in the case base mainly contains the case identification, problem description, and associated solution. In contrast with the Case-Base, the package of the Domain-Object in the bottom layer is used to represent individual domain concepts, which has to be customized according to each particular domain. In a given domain, The Domain-Object denotes a collection of physical elements in the AHU, such as dampers and fans. Moreover, in the middle layer, we defined a package called Object-Network, which is able to make the connection between the package of the Case-Base and Domain-Object. The problem description in cases has a composition relation with one Object-

Network. The fundamental entities in Object-Network include the nodes and edges. The nodes map directly to the objects defined in the Domain-Object. The Object-Network in each case has a one-to-more relation represented as an edge that relates to the nodes. The nodes are reachable from the edges. Each edge in the Object-Network is associated with a numerical weight value, which expresses the importance of this node (or this domain object) as a member to the particular case.

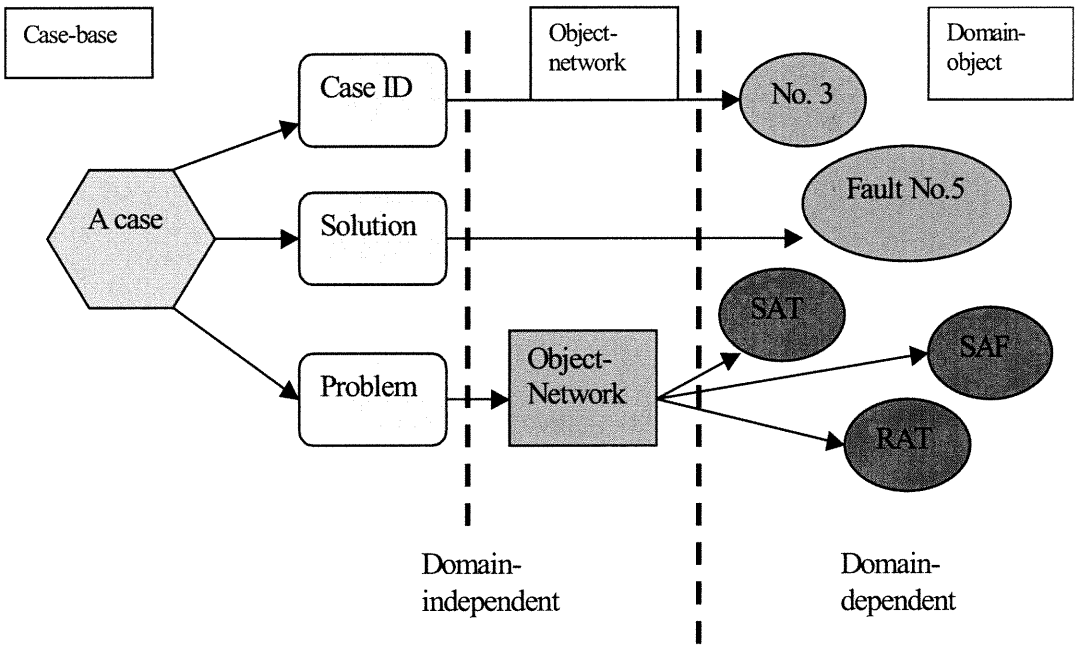


Figure 5 - 3 Structure of case

Clearly, the Object-Network in the CBR framework plays a role as a bridge in that it links up the domain-specific and domain-independent knowledge. The packages of the Case-Base and Object-Network are considered to be reusable in the CBR formwork, since there is nothing concerning with any particular domain knowledge but a basic skeleton of CBR representing structure. It could only be necessary to redesign the part of Domain-Object if changing a domain problem. The design of reasoning processes could also be domain-independent, since it only refers to the Object-Network.

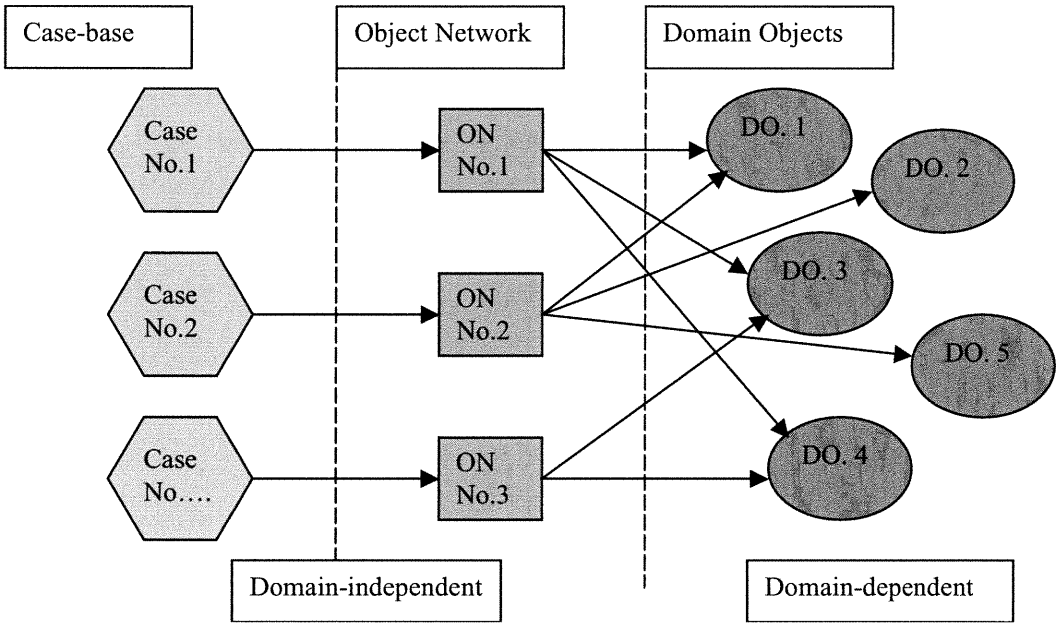


Figure 5 - 4 Case representations

In Figure 5 – 4, the structure of the entire case base is illustrated. With this case representing structure, it would be abstract enough to achieve a higher degree of reusability and concrete enough to represent domain knowledge more accurately. The advantages of this case representation are summarized as below:

- The structure of the case base is flexible and extensible, since any domain concepts can be connected to the cases.

- The case base can be tuned up during execution, since the modification of cases is so simple; it is only concerned with changing the definition of the edges in the Object-Network.

- The design of reasoning processes could be domain-independent, since it only refers to the Object-Network.

- The system can handle incomplete problem queries, since the associated weight is sensitive to the missing and noisy members.

- In addition, the design concepts of the case representation remain simple and clear. It is easy to perform learning without changing the program codes.

5.4.2. Reasoning Process

After the designing of case representation, the emphasis of the phase of development is now on how to operate the knowledge that is encoded in the case representation so that a reasonable result can be drawn. To satisfy the basis of case reasoning, several popular similarity functions are integrated into the classes of Distances and Algo, such as the classical nearest neighbor, k-nearest neighbor, and so on. In the way of case representation that we designed, it allows the methods of the reasoning process to be domain-independent in the CBR framework, and it still keeps open to add new reasoning methods.

The reasoning process starts from the appearance of a new problem. To speed up the case retrieval and reduce the calculation, a simplified similarity measure is used ahead. The equation is:

$$LocalSim(O_P, O_C) = \begin{cases} 1 & if \quad O_P = O_C \\ 0 & if \quad O_P \neq O_C \end{cases}$$

Where O_P and O_C indicate a domain object contained in new problem and case, the return value of function is one if the type of two domain objects are identical, otherwise, the return value is zero. Clearly, this simplify measure does not offer much information about similarity, but it gives a quick look at the case base to check which case holds the same domain objects as much as the new problem. A case quickly becomes disqualified for the further reasoning process as soon as there are no domain objects contained in that case that match the new problem. The qualified cases will continue to calculate the similarity measure by the k-

nearest neighbor. Details about k-nearest neighbor technology will be explained in a later chapter. This calculation is repeated for every qualified case in the case base and normalized to fall within a range from zero to one where zero is totally different and one is an exact match. If the value assigned to a given case by the k-nearest neighbor exceeds a lower bound of diagnosis threshold that it locally defined in case base, then this case is said to be good as a solution. A solution list including all the cases of value that exceed the diagnosis threshold is ready for voting on by human experts either in the positive or negative. If the result is positive, the reasoning process will stop here with success. If not, the process will ask human experts to repair the case and add it to the case base.

Finally, without providing basic functions supporting such as File I/O and common algorithms, the framework is incomplete, so that we implement a set of basic functions in the Utility class.

Chapter 6.

The Implementation of a CBR Diagnosis System

6.1. Overview

In this chapter, the implementation of a CBR diagnosis system is presented. Section 6.2 introduces the package of the Case-Base and shows the definition of an individual case. Section 6.3 explains the package of the Domain-Object, which focuses on the issues of domain modeling. Section 6.4 introduces the package of the Object-Network, which is the core component of case representation. Section 6.5 illustrates similarity functions, including Minkowsky metric and Euclidean distance. Section 6.6 discusses the evaluation and learning issues in our approach.

6.2. Individual Cases

In the case representation that we proposed, every individual case contained in the case base is entitled by three interrelated items: a case identification, a problem description, and an associated diagnostic solution. The visualization of an individual case is shown in Figure 6-1.

- The case identification possesses an object called Case ID, which contains a unique label to distinguish one case from others.

- The problem description is an abstract java class, which has a link to the Object-Network. The Object-Network consists of a set of nodes and edges. A node denotes a particular domain object in the AHU, such as a cooling coil valve. The notes are reachable from the edges.

- The solution is also an abstract java class, which contains a unique fault number indicating what element has failed. For convenience, the fault number is consistent with the definition of the expert detection system. Such a fault NO.5 in two systems has the same meaning that says "sensor failure in return air temperature".

Each individual case is saved in a dimensional vector space. The i^{th} component of the vector represents case C_i. The whole of the case based is currently saved in a sequence file. It is also possible to change the case base storage into a database, if required.

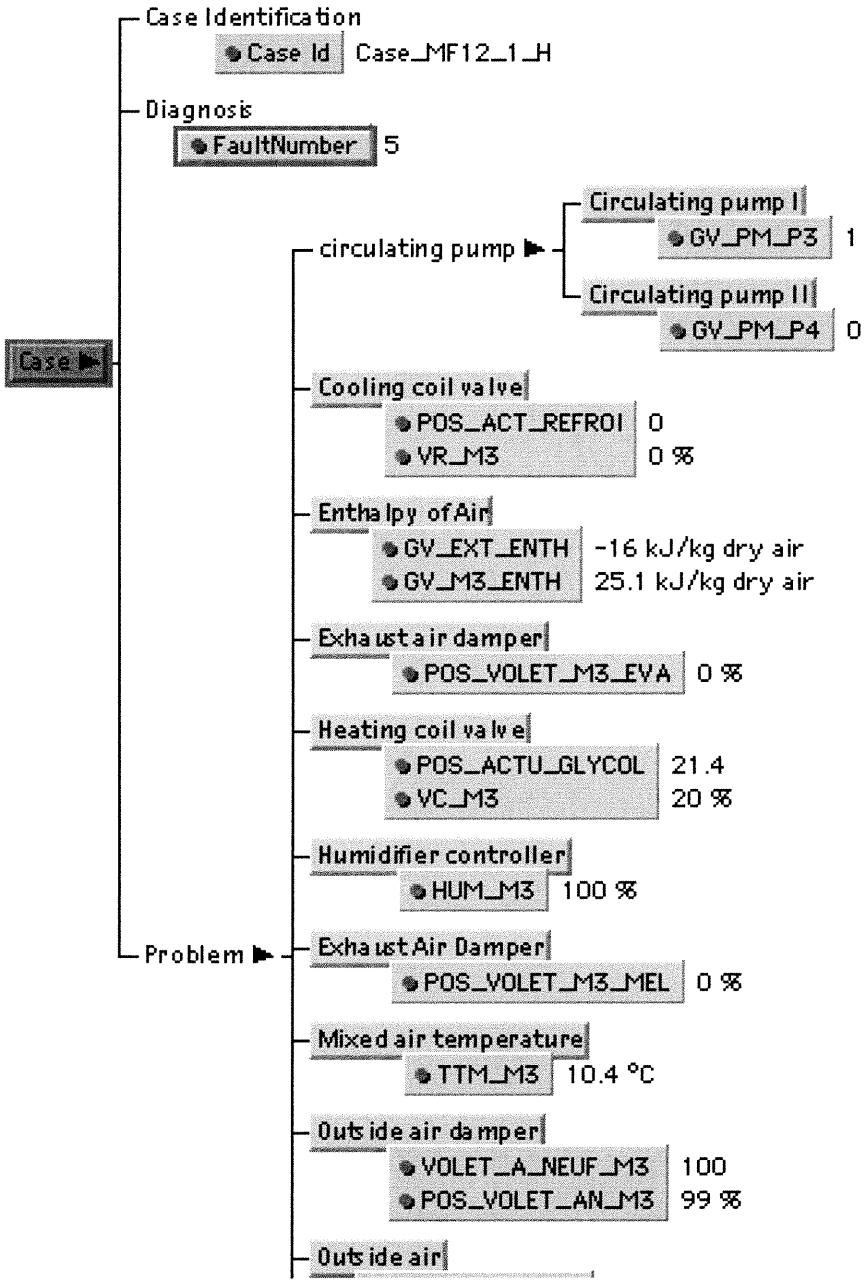


Figure 6 - 1 the visualization of case structure

6.3. Domain Modeling

At this level we work with specific domain modeling, which is the activity of identifying and conceptualizing domain concepts and relationships, and turning them into a computer understanding language. In general, the process of domain modeling based on object-oriented techniques may be broken into the following two steps: finding objects and defining abstraction to classify those objects.

To find an object, or to isolate an object from others is the hardest part of object-oriented analyses and designs. We need to figure out what knowledge exists in a domain and how it can be used in problem solving. The objects in domain modeling can be defined as physical items or abstract concepts. Since the purpose of a CBR diagnosis system is to find the cause of physical failures, all the objects that we defined naturally come from the physical components of the AHU or abstracted definitions of them.

The relationship issue that is not yet discussed is how these objects should be organized and how they relate to each other. Object-oriented modeling techniques offer the mechanism that objects can be organized into a hierarchy so that those objects can together form a structured representation. This hierarchy serves to organize similar objects into groups named classes and then to organize similar classes into super classes. The objects at any level of the hierarchy inherit all the attributes and methods of the higher-level objects. This kind of inheritance mechanism makes it possible to reduce the amount of information that has to be stored at each level in the hierarchy.

In this thesis, all domain concepts and relationships in the AHU shown in Figure 6-2 are defined in the package of the Domain-Object, which contains the declarations of domain states, relations, properties and values.

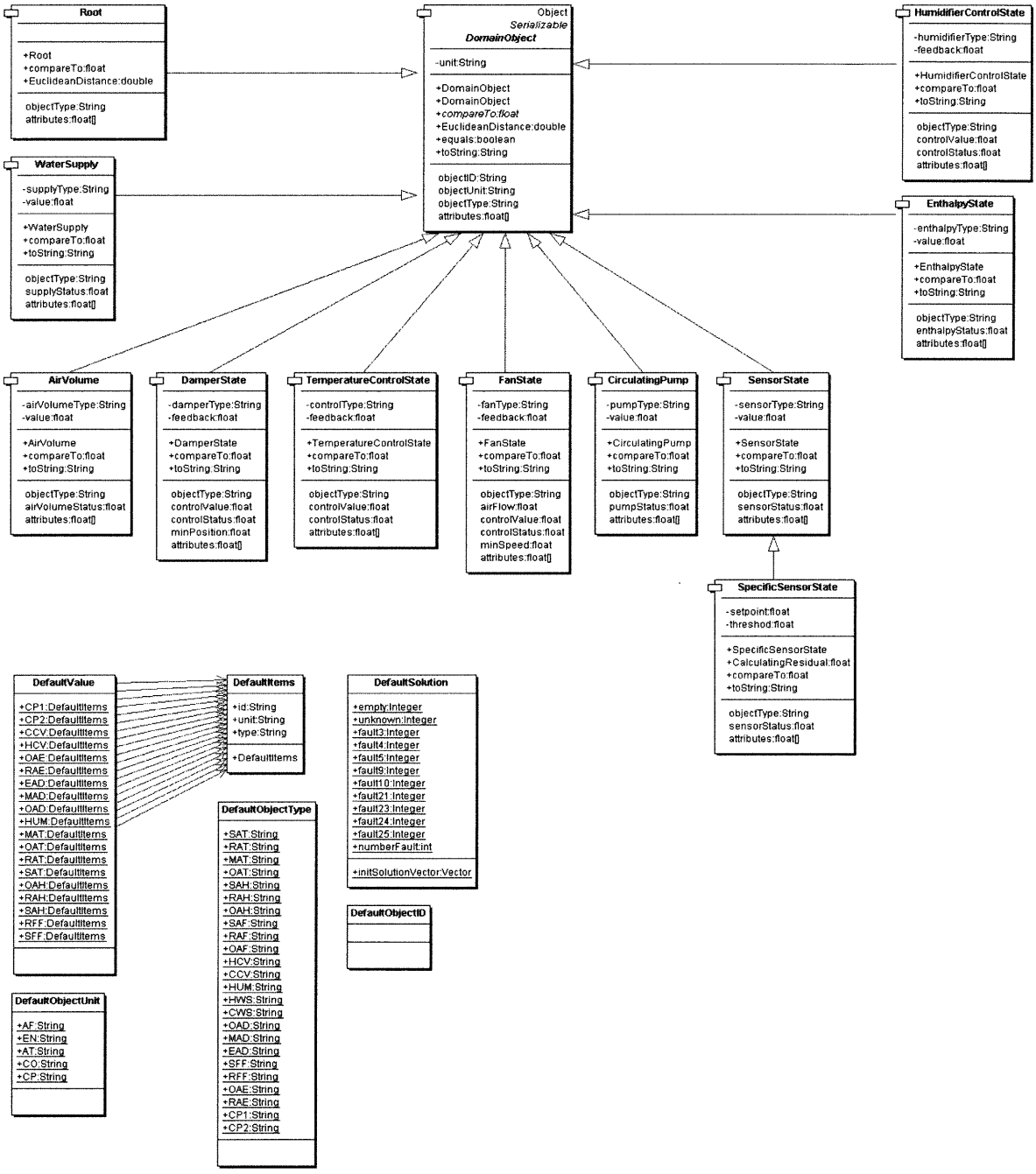


Figure 6 - 2 Domain objects modeling

- The domain states are used to represent similar concepts of knowledge that exist individually in the AHU, such as sensor-State, damper-State, etc; or a few higher level abstracted entities, such as the class of Domain-

Object, which does not represent any specific knowledge but can be inherited by other classes. The domain states have a set of attributes declarations and methods definitions. Every object in the AHU is an instance of some domain states. For instance, in Table 6-1, the sensors of return and mixing air temperatures are separately two physical components in the AHU. But since two types of sensors have same characters and functionalities, so the two objects are the instance of sensor-States but different types. All methods on an object are defined in the domain states of that object, or in the super class of domain states.

- The relations describe how the domain states are mutually involved. The relations can range the domain states into an inheritance hierarchy. This inheritance makes it possible for the domain states to share attributes and messages. The class of Domain-Object is on the top of this hierarchy. All other domain states are the subclass of the Domain-Object. During execution, the search for an attribute begins at some level of the hierarchy and proceeds to the top. Hence, an attribute at a lower level can be hidden at a higher-level object. For instance, in Figure 6-3 and Table 6-2, Object-id is an attribute defined in the class of Domain-Object; every object that is an instance of any domain states can access this attribute by a pre-defined method.

- The messages describe the interaction between objects and the resulting changes. The messages in object-oriented concepts can be looked at as functions or procedures in conventional programming. Usually, a message is a request to a specific object to invoke one of its methods to perform certain operations. The same as attributes, the message at a lower level can also be hidden at a higher-level object. In addition, the message in the higher-level class can be defined in abstract form, which forces all subclasses to specialize it in order to provide coherent behaviors for the overall system. For instance, in Table 6-1 and Table 6-2, a message called

compare-To that is used to calculate the similarity between two objects is an abstract class defined in the class of Domain-Object. Every object that is an instance of any domain states has to rewrite this function according to its own attributes.

- The attributes and default value reflect the feature of domain concepts. How many attributes should be declared? It depends on the functional requirements of systems. It will not be necessary to create an exhaustive detail of each object.

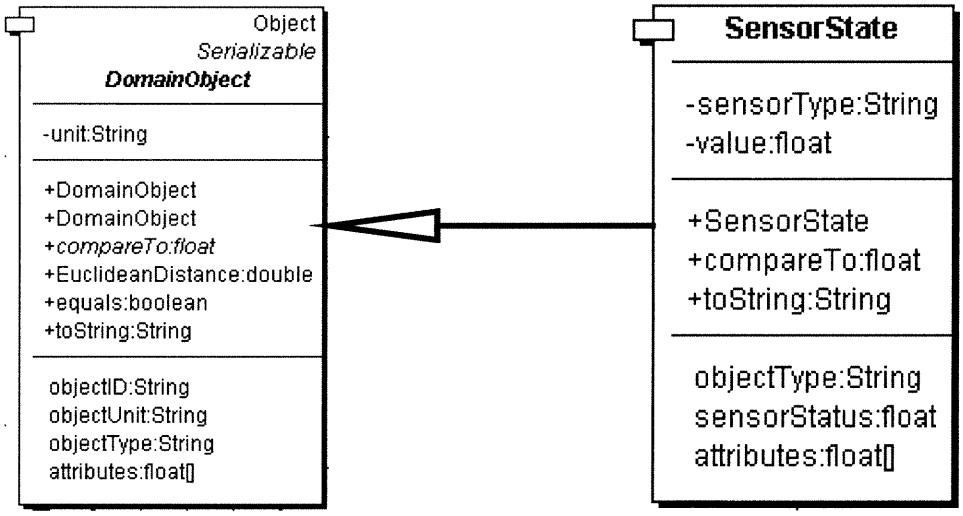


Figure 6 - 3 Domain-Object & Sensor-State

public abstract class DomainObject extends Object implements Serializable		
Hierarchy	java.lang.Object | +--domainObject.DomainObject	
Direct Known Subclasses	AirVolume, CirculatingPump, DamperState, EnthalpyState, FanState, HumidifierControlState, Root, SensorState, TemperatureControlState, WaterSupply	
Methods inherited from class java.lang.Object	getClass, hashCode, notify, notifyAll, wait, wait, wait	
Fields	**ObjectID** **Unit**	String String
Constructor detail	**DomainObject**	public DomainObject()
	DomainObject	public DomainObject(String id, String objectUnit)
Method detail	**CompareTo**	public abstract float compareTo(DomainObject ob) /*Compare two domainObject. Returns: a float of the comparaison */
	Equals	public boolean equals(java.lang.Object domainObject2) /*Compare "this" with another domain object. This method check only if two domain object are the same type or not. Returns: True if two object networks are the same, false otherwise. */
	EuclideanDistance	public double EuclideanDistance(DomainObject other) /*Determine the distance between two domain objects with a known weight. Parameters: another domain object Returns: a distance = sqrt [sum(distanceAttribut^2 * weight)] */
	GetAttributes	public abstract float[] getAttributes() /*Get all attributes in an array. Returns: an array with all attributes */
	GetObjectID	public String getObjectID() /*Get the id of the domain object. Returns: the ObjectId */
	GetObjectType	public abstract String getObjectType() /*Get the type of a domain object. Returns: the type */
	GetObjectUnit	public String getObjectUnit() /*Get the unit of the domain object. Returns: the unit */
	ToString	public String toString() /*Convert this object into a String Returns: a textual description of this node */
Association Links	to Class java.lang.String	

Table 6 - 1 Domain-Object class

public class SensorState extends DomainObject Sensor type: 1.Supply air temperature 2.Return air temperature 3.mixed air temperature 4.outside air temperature 5.Supply air humidity 6.Return air humidity 7.outside air humidity		
Hierarchy	java.lang.Object | +--domainObject.DomainObject | +--domainObject.SensorState	
Direct Known Subclasses	SpecificSensorState	
Methods inherited from class domainObject.DomainObject	Equals, EuclideanDistance, getObjectID, getObjectUnit	
Methods inherited from class java.lang.Object	getClass, hashCode, notify, notifyAll, wait, wait, wait	
Fields	**SensorType**	String
	Value	Float
Constructor detail	**SensorState**	public sensorState(String id, String unit, String type, float status)
Method detail	**CompareTo**	public abstract float compareTo(DomainObject ob) Compare two domainObject. /*Returns: a float of the comparaison */
	GetAttributes	public float[] getAttributes() /*Get all attributes in an array. Returns: an array with all attributes */
	GetObjectType	public abstract String getObjectType() /*Get the type of a domain object. Returns: the type */
	getSensorStatus	public float getSensorStatus()
	ToString	public String toString() /*Convert this object into a String Returns: a textual description of this node */
Association Links	to Class java.lang.String	

Table 6 - 2 Sensor-State class

6.4. Object-Network

Object-Network is a bridge that connects up a case to several domain objects. Figure 6 - 4 illustrates the package of Object-Network defined here. In Object-Network, there are two basic primitives; namely, nodes and edges. The nodes may represent any objects that declared in the package of Domain-Objects, such as a mixing air damper or supply air fan. An Object-Network may have a set of nodes, in which they, all together, denote a particular case. The nodes are reachable from the edges. Each edge is assigned to a weight that the value of weight that might have come from experience or experiments. This weight is used to express the importance of this object as a member for the case.

An Object-Network is used to represent the more important objects or characters that can be compared in different cases. The case retrieval process is performed by first calculating the distance between two nodes in different cases and then summing up the distances of all nodes in order to indicate the similarity of two entire cases. Figure 6-4 shows the design of the Object-Network package. A bunch of interface and events are declared in the package.

Events are treated to take into account case structural modifications. The events can be hooked with the graphic interface so that any modifications of the case can be made, such as changing a value or redrawing a line on computer screen.

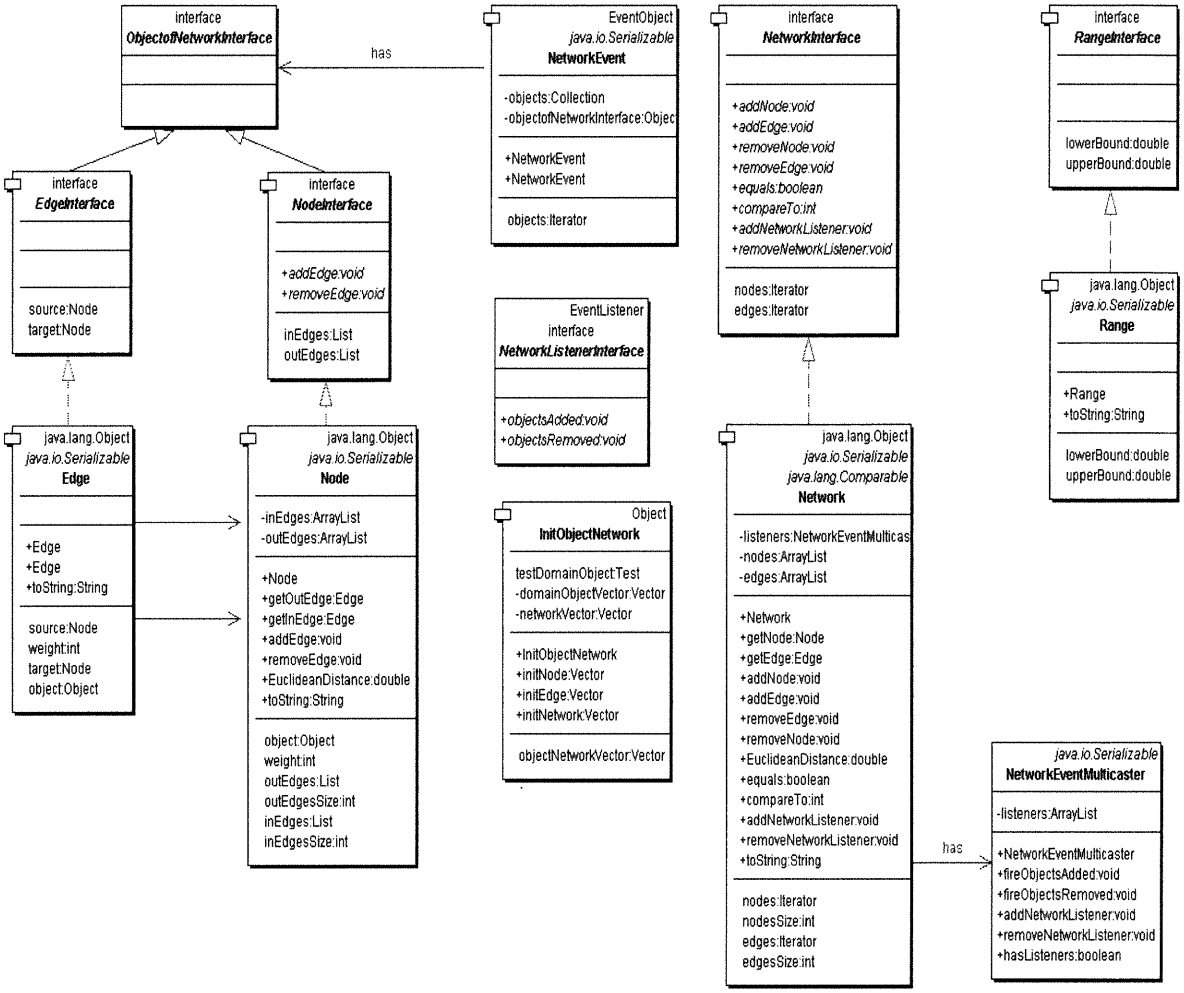


Figure 6 - 4 Package of Object-Network

6.5. *Similarity Measure*

To be able to find cases that are similar to the current problem, we need a similarity measure. Usually, the similarity measure between a new problem and a case in the case base is computed in a bottom up fashion: first at an objects level, a local similarity measure determines the similarity between the same attributes in two objects. The local similarity measure can be multiplied by a weight factor. Then, at the cases level, a global similarity measure determines the similarity between two cases by summing up the local seminaries of the belonging objects.

The k-nearest neighbor is perhaps the most widely used similarity measure algorithm in CBR. The k-nearest neighbor is a conventional algorithm that provides good performance for optimal values of k. In the k-nearest neighbor algorithm, it is supposed to have k pre-defined cases that can be looked at as patterns. The k-nearest neighbor is able to handle the $k+1^{th}$ case that can be classified to the k pre-defined cases. The assigned case assures that the average distance of every attribute in the case is in a minimum to the k+1 case. The minimum distance refers to the maximum similarity between the two cases.

A major problem of the simple approach of k-nearest neighbor is that it uses all attributes in computing distances without weight factor. In many cases, only a few attributes are truly relevant to the classification task. A possible way to overcome this problem is to assign weights for different attributes. The practical implementation used in our approach of calculating the similarity is based on the Minkowski metric shown in Figure 6-5. The equation of the Minkowsky metric is defined as [37]:

$$Minkowshy(P,C) = \sqrt[r]{\sum_{i=1}^{n} W_i \times \left|P_i - C_i\right|^r}$$

Where P and C are the new problem and case, whose similarity is computed; W_i is the weight factor that assigned to every attribute; n is the number of attributes in the new problem and case. P_i and C_i represent the value of the i^{th} attribute of the new problem and case respectively. Where r in this equation can be three different values where each of them refers to different distance calculations: Hamming distance ($r = 1$), Euclidean distance ($r = 2$), and Cubic distance ($r = 3$).

The Euclidean distance is one of the most immediate and most frequently used similarity measures in CBR. It is especially suitable to the calculation of the distance between two vectors where it is that we store the new problem and cases, so that we declared a specialized abstract class for Euclidean distance shown in

Figure 6-5. This measure is defined as the square root of the sum of the squared differences between the two vectors, of which one belongs to the new problem, and another belongs to a completed case in the case base.

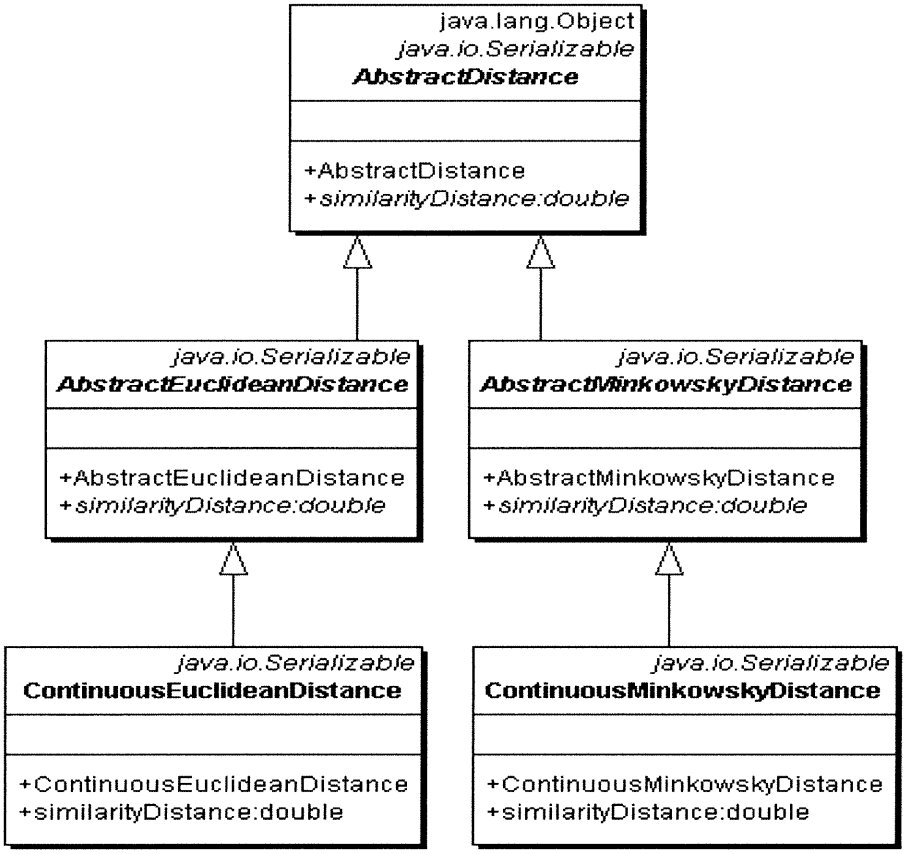


Figure 6 - 5 Similarity measure

The Euclidean distance is the shortest path between two points. For instance, a geometric interpretation of the Euclidean distance between the point *P1* and *P2* is explained in the Figure 6-6. The weight for both *P1* and *P2* are supposed to be one in this example.

$$EuclideanDis1(P1,P2) = \sqrt{(5-2)^2 + (5-1)^2} = 4$$

In addition, the weight can be adjusted in the CBR diagnosis system. An appropriate weight setting enables us to increase the accuracy rate of results in the system. The initial values of weights can be estimated from statistical experiments and experiences. The following example in Figure 6-7 shows that the weight can affect the result of similarity.

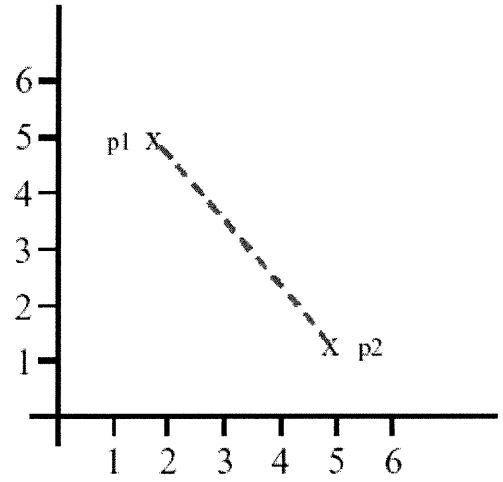


Figure 6 - 6 Euclidean distances

$$EuclideanDis2(P1,P2) = \sqrt{0.2 \times (5-2)^2 + 0.8 \times (5-1)^2} = 2$$

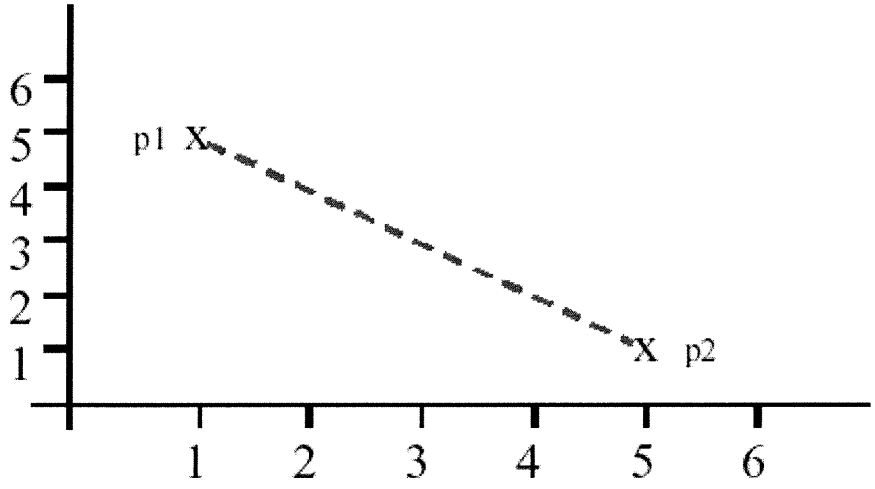


Figure 6 - 7 Weight factor

As Figure 6-7 shows, the attribute in the horizontal axis has a greater weight (0.8) than the one in the vertical axis (0.2), so that the Euclidean distance gives more influence to the one in horizontal axis in the computation of the distance.

One weakness of the Euclidean distance function is that one of the input attributes has a relatively large range, and then it can overpower the other attributes. For the same example as above, if P1 can have values from 1 to 1000, and P2 can only have values from 1 to 10, as a result, the influence of P1 on the Euclidean distance will overwhelm the influence of P2. Therefore, the Euclidean distance needs to be normalized by dividing the distance for each attribute by the range of that attribute, so that the Euclidean distance for each attribute is in the approximate range of 0..1.

In Table 6-4, shows the 5 nearest cases given by the retrieval phase when testing on an artificial input situation. The similarity function used is Euclidean distance. Detailed information about sample cases can be found in appendix A.

Nearest cases	**Case id**	**Fault diagnosis**	**Similarity**
1	Case_MF17_1_H	F9	0.95
2	Case_MF39_1_H	F9	0.87
3	Case_MF6_1_H	F21	0.76
4	Case_MF18_1_H	F5	0.42
5	Case_MF5_1_DM2	F9	0.26

Table 6 - 3 Experiment result

6.6. *Evaluating & Learning*

In this version of the CBR diagnosis system, the evaluating and learning mechanisms rely heavily on human experts themselves. There is no method that

uses any general knowledge model to achieve evaluation and learning automatically.

A list of match cases like the one shown in Table 6-4 is produced by a retrieval process. Human experts can look through this list and examine the detailed record of each past case, using their experiences and judgments to select the most appropriate of the matches. Then, after fixing the failures of equipments, human experts might vote for this case in the positive or negative. The process of evaluation at this moment is only used to be an additional filter that reduces the growth of the case base and improves the accuracy of case base. If the CBR diagnosis system fails to find solution, human experts are asked to create a new case that describes the current problem and adds it to the case base by hand. To avoid the size of the case base growing unexpectedly, cases of forgetting can be realized based on deleting not recently used or the least used cases. The forgetting function has not been realized in this version yet.

Chapter 7.

Conclusions and Future Work

7.1. Conclusions

This thesis discussed a system that uses CBR as a problem solving methodology to deal with the diagnostic problems for the AHU. Several conclusions that can be drawn from this work are summarized below:

- AHU systems have played an important role in each modern building; however, faults occurring in AHU are common. It is not an easy job to determine the causes of faults. A certain level of the experience and knowledge of AHU is required. Thus, it is valuable to build a knowledge-based system to help building operators automate the fault diagnosis process.

- As the behavior of AHU is dynamic and poorly understood, it is difficult to formalize general rules to solve diagnostic problems. In contrast, by

using CBR techniques, a set of past experiences about AHU can be stored in a case base to suggest solutions.

- Since the burdens of knowledge acquisition issues have been shed, thus, CBR systems can be built more easily than traditional knowledge-based systems.

- CBR systems can integrate knowledge reasoning mechanisms, knowledge storage and learning into one platform. Therefore, a CBR system is a kind of autonomous system that can improve performance over time.

- A conceptual CBR framework is built to allow certain flexibilities and generalities. Therefore, this CBR framework can possibly be reused and expanded to later CBR applications without changing the entire system structure.

- An object-oriented case representation is developed in a way that is domain-independent; thus, it is able to represent knowledge in various kinds of domains instead of a particular domain.

7.2. Contributions and Future Work

We think that the main contributions of our approach are: First, a CBR diagnosis system is built by using CBR techniques to solve one of the sophisticated problems in HVAC. Secondly, a generic CBR framework is built that allows later CBR applications to reuse and share its designs and implementations.

However, this system is currently in a prototype stage. The testing results so far are encouraging, but it is still far from our ultimate objectives. The following future work can be considered.

- **System integration issues:** an integration of CBR and other reasoning systems is a considerable way from making CBR applications more competent and robust to solve real problems in the world.

- **Case representation issues:** the cases in a case base might be extended to be able to cover all relevant knowledge types, including general and specific knowledge in domains.

- **Case base management issues:** an individual case base management system is necessary to facilitate the processes of case collections and case maintenances.

- **Evaluation issues:** the evaluations of proposed solutions should be done in a completely automatic way. Deeper domain knowledge can be used to justify the solutions so that the quality of results can gradually be improved.

- **Learning issues:** a practical learning method is needed to modify a rejected solution dynamically during the time of the CBR reasoning process, so that CBR systems are able to adapt to an evolving environment.

Appendix A

Sample cases

Case 1~3:

Attributes	Case_MF12_1_H	Case_MF16_1_H	Case_MF17_1_H
Circulating pump status	1	0	0
Circulating pump status	0	1	1
Cooling coil status	0	0	85.4
Cooling coil valve control	0	0	42.9
Enthalpy	-16	-3	-9
Enthalpy	25.1	22.1	22.7
Exhaust air damper position status	0	99.5	0
Heating coil status	21.4	87.3	64.3
Heating coil valve control	20	89.6	64
Humidifier control	100	100	100
Mixed air damper position status	0	0	43.4
Mixed air temperature	10.4	95.4	11.9
Outside air damper control	100	100	100
Outside air damper position status	99	99	99
Outside air humidity	61.2	54.6	52.9
Outside air temperature	-13	-12	-5
Return air humidity	31.4	32.2	32.2
Return air humidity set point	40	40	220.9
Return air temperature	24.5	20.7	21.5
Return fan air flow	215.4	193.8	229.7
Supply air humidity	34.1	27.4	25.4
Supply air temperature	15.3	21.8	21.4
Supply air temperature set point	14.8	21	21
Supply fan air flow	350.8	324.4	350.8
Supply fan status	1	1	1
Fault Number	**5**	**9**	**9**

Case 4~6:

Attributes	Case_MF18_1_H	Case_MF39_1_H	Case_MF5_1_DM2
Circulating pump status	0	0	0
Circulating pump status	1	1	0
Cooling coil status	0	79.9	35.8
Cooling coil valve control	0	82.4	87.9
Enthalpy	-2	0.1	23
Enthalpy	22.5	22.9	26.3
Exhaust air damper position status	0	0	49.5
Heating coil status	64.4	11.7	0.6
Heating coil valve control	65.4	2	0
Humidifier control	100	100	0
Mixed air damper position status	0	0	0
Mixed air temperature	11.9	13.5	25.1
Outside air damper control	100	100	100
Outside air damper position status	99	99	0
Outside air humidity	54.1	51.9	58
Outside air temperature	-6	-11	18.2
Return air humidity	32.3	32.4	47.8
Return air humidity set point	40	40	30
Return air temperature	21.2	21.9	23.4
Return fan air flow	208.2	350.8	215.4
Supply air humidity	25.9	23.8	42
Supply air temperature	21	27	23.1
Supply air temperature set point	21	21	17.8
Supply fan air flow	342	377.1	342
Supply fan status	1	1	1
Fault Number	**10**	**21**	**4**

Case 7~9:

Attributes	Case_MF5_2_DM2	Case_MF5_3_DM2	Case_MF5_4_DM2
Circulating pump status	0	0	0
Circulating pump status	0	0	0
Cooling coil status	0	0	0.5
Cooling coil valve control	100	0	0
Enthalpy	25.5	25.4	23.8
Enthalpy	27	28.6	25.5
Exhaust air damper position status	12.6	0	0
Heating coil status	0.6	0.5	0.6
Heating coil valve control	0	0	0
Humidifier control	0	0	0
Mixed air damper position status	0	0	0
Mixed air temperature	22.7	23.7	24.2
Outside air damper control	100	100	100
Outside air damper position status	0	0	0
Outside air humidity	44	79.8	45.9
Outside air temperature	22.4	17.5	20.5
Return air humidity	46.3	53.3	44.1
Return air humidity set point	30	30	30
Return air temperature	23.9	23.4	22.5
Return fan air flow	136.4	78.9	172.3
Supply air humidity	41.3	48.6	43.2
Supply air temperature	24.4	24.7	25
Supply air temperature set point	16.3	17.3	20
Supply fan air flow	0	0	0
Supply fan status	0	0	0
Fault Number	**4**	**3**	**24**

Case 10~11:

Attributes	Case_MF6_1_H	Case_MF6_2_H
Circulating pump status	1	0
Circulating pump status	0	1
Cooling coil status	0	0
Cooling coil valve control	0	0
Enthalpy	-13	-7
Enthalpy	23.2	23.4
Exhaust air damper position status	0	0
Heating coil status	99	42
Heating coil valve control	100	41.6
Humidifier control	100	100
Mixed air damper position status	0	0
Mixed air temperature	12.7	12.7
Outside air damper control	100	100
Outside air damper position status	99	99
Outside air humidity	60.4	57.7
Outside air temperature	-17	-14
Return air humidity	31.4	31.8
Return air humidity set point	40	40
Return air temperature	22.2	22.2
Return fan air flow	272.8	258.4
Supply air humidity	24.2	30.8
Supply air temperature	21.1	20.5
Supply air temperature set point	20.8	20.3
Supply fan air flow	412.1	377.1
Supply fan status	1	1
Fault Number	**3**	**3**

Bibliography

[1] M. A. Carrico, J. E. Girard, and J. P. Jones, *Building Knowledge Systems: Developing and Managing Rule-Based Applications.* McGraw-Hill Book Company, New York 10023, 1989.

[2] P. Haves and S. K. Khalsa, *Model-Based Performance Monitoring: Review of Diagnostic Methods and Chiller Case Study.* Lawrence Berkeley National Laboratory. California, 2000.

[3] CEDRL, http://cerdrl.mets.nrcan.gc.ca

[4] Delta Control, http://www.deltacontrols.com.

[5] A. Aamodt and E. Plaza, *Case-Based Reasoning: Foundation Issues: Methodological Variations, and System Approaches.* AI Communications, pp. 39-59, 1994.

[6] M. E. Fayad, D. C. Schmidt, and R. E. Johnson, *Building Application Frameworks.* Addison-Wesley Pub Co, 1st edition, 1999.

[7] M. Greenwell, *Knowledge Engineering For Expert Systems.* Ellis Horwood Limited, England, 1988.

[8] E. Turban, *Expert Systems And Applied Artificial Intelligence*. Prentice Hall business publishing, pp. 117-139, 1993.

[9] S. J. Russell and P. Norvig, *Artificial intelligence: a modern approach.* Prentice Hall, Upper Saddle River, New Jersey, 1995.

[10] G. F. Luger and W. A. Stubblefield, *Artificial intelligence: structures and strategies for complex problem solving*. The Benjamin/Cummings Publishing Company, Inc, Redwood City, California, 1994.

[11] A. Aamodt, *A Knowledge-Intensive Approach to Problem Solving and Sustained Learning*, PhD dissertation, University of Trondheim, Norwegian Institute of Technology, 1991.

[12] A. Aamodt, *Case-Based Reasoning: An introduction*. University of Trondheim, Norway, 1995.

[13] A. Aamodt and E. Plaza, *Case-Based Reasoning: Foundation Issues: Methodological Variations, and System Approaches.* AI Communications, pp39-59, 1994.

[14] D. B. Leake, *Case-Based Reasoning, Experiences, Lessons & Future Directions.* AAAI Press, 1996.

[15] R. Schank, *Dynamic memory: a theory of reminding and learning in computers and people*. Cambridge University Press. 1982.

[16] J. Kolodner, *Maintaining organization in a dynamic long-term memory.* Cognitive Science, Vol.7, pp. 243-280. 1983.

[17] R. L. Simpson, *A computer model of case-based reasoning in problem solving: An investigation in the domain of dispute mediation.* Technical Report GIT-85/18. Georgia Institute of Technology. 1985.

[18] K. Sycara, *Using case-based reasoning for plan adaptation and repair.* Proceedings Case-Based Reasoning Workshop, DARPA, Clearwater Beach, Florida, pp. 425-434. 1988.

[19] K. J. Hammond, *Case-Based Planning*. Academic Press, 1989.

[20] P. Koton, *Using experience in learning and problem solving*. Massachusetts Institute of technology, Laboratory of Computer Science. Mit/LCS/TR-441. 1989.

[21] T. R. Hinrichs, *Problem solving in open worlds*. Lawrence Erlbaum Associates. 1992.

[22] Ray Bareiss, *PROTOS: A unified approach to concept representation, classification and learning.* Technical report ai88-83, University of Texas at Austin, 1989.

[23] E. Rissland, *Example in legal reasoning: Legal hypothetical*. Proceedings of the Eighth International Joint Conference on Artificial Intelligence, IJCAI, Karlsruhe. 1983.

[24] ReMind, http://www.ai-cbr.org/ tools/remind.html.

[25] Kate, http://www.ai-cbr.org/tools/acknosoft.html.

[26] CBR-Works, http://www.cbr-web.org/CBR-Web/CBR-Works/.

[27] CBR – tools, http://www-sop.inria.fr/aid/cbrtools.

[28] M. M. Richter and S. Wess, *Similarity, Uncertainty and Case-Based Reasoning in PATDEX.* University of Kaiserslautern, 1991.

[29] B. Lopez and E. Plaza, *Case-Based planning for medical diagnosis*. 7th International Symposium, 1990

[30] A. Aamodt, *A Computation Model of Knowledge-Intensive Learning and Problem Solving.* University of Trondheim, Norway, 1995.

[31] I. Watson, *Applying Case-Based Reasoning: Techniques for Enterprise Systems.* Morgan Kaufmann Publishers Inc, 1997.

[32] J. Kolodner, *Case-Based Reasoning.* Morgan Kaufmann Publishers Inc, 1993.

[33] I. Watson and F. Marir, *Case-Based Reasoning: A Review.* Cambridge University Press, The Knowledge Engineering Review, Volume 9, and No. 4: pp. 355-381, 1994.

[34] Draft, *Fault Detection and Diagnosis Tool.* Document of System Requirements, 2001.

[35] P. Carling, *Automated Fault Detection in Custom-Designed HVAC System.* Royal Institute of Technology Building Services Engineering, Bulletin no.54, 2000.

[36] P. Stevens and R.J. Pooley, Using UML: *Software Engineering with Object and Components.* Addison-Wesley and Personal Education, 2000.

[37] B. G. Batchelor, *Pattern Recognition: Ideas in Practice.* New York: Plenum Press 1978.

[38] C. K. Riesbeck and R. S. Schnk, *Inside case-based Reasoning.* Erlbaum, Northvale, New York, 1989.