INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

A Case-Based Reasoning Approach to the Designing of Building Envelopes

Serban Iliescu

A Thesis
in
The Department
of
Building, Civil and Environmental Engineering
Faculty of Engineering and Computer Science

Presented in Partial Fulfillment of the Requirements
For the Degree of Doctor of Philosophy at
Concordia University
Montreal, Quebec, Canada

ABSTRACT

A Case-Based Reasoning Approach To Building Envelope Design

Serban Iliescu

Building-envelope design is an information-intensive process that requires experiential knowledge. Confronted with such a process, a human expert adds to well-known domain knowledge his own experience, or the experience of others, to support his reasoning process and guide him in typical situations.

The problem-solving paradigm where reasoning is supported by reusing past experiences is called *Case-Based Reasoning (CBR)*, and it was added to the *Artificial Intelligence (AI)* methodology following research in cognitive psychology. Instead of relying solely on general knowledge of a problem domain, or making associations along generalized relationships between problem descriptors and conclusions, CBR is able to utilize the specific knowledge of previous experienced problem situations called *cases*. CBR is a technology that solves problem by storing, retrieving, and adapting past cases. CBR systems have been proposed as an alternative to rule-based systems whenever the knowledge engineering process of eliciting rules is difficult or unmanageable. Instead, many experiences (or cases) with solutions, warnings, plans, and so forth are collected and new situations are related to a stored recollection of these past cases. New solutions are adapted from the old ones.

Research in *Knowledge-Based Expert Systems (KBES)* for building-envelope design has shown a similar trend. While computerized assistance was imposed by the large amount of data

to be processed, the success of rule-based implementations was hampered by the lack of abstract domain knowledge. Such fields where most of the knowledge is based on experience are often labeled as "*weak theory domains*," and they are prime candidates for adopting a CBR approach.

This thesis proposes a CBR framework for selecting the construction alternatives during the preliminary stage of the building-envelope design process. The methodology presented aims to find the most suitable design for a new building envelope from a library of prototypical building cases and adapts it to meet the requirements of ASHRAE Standard 90.1/1989 for energy efficient building design. The study outlines the potential benefits of using CBR technology and the key issues encountered while attempting to define the CBR model for building-envelope design. Developing a hierarchy of building-envelope components identifies cases and features for design. The envelope design problem is solved through decomposition, and by combining case-based and rule-based reasoning methods. In searching for a best match to achieve a higher degree of case filtering, a connection between case-based reasoning and *Artificial Neural Networks (ANN)* is proposed. An ANN-based filtering mechanism is designed to improve the quality of case-matching outcome while enforcing the economy of case representation.

The framework proposed by this research has been implemented into the CRED software system demonstrating the feasibility and advantages of using CBR methodology for building envelope design. CRED blends several AI techniques (such as ANN, CBR and KBES) while aiming to offer expert assistance to building design professionals for browsing and selecting building-envelope alternatives.

ACKNOWLEDGMENTS

There several people who deserve my thanks, whose input, advice or support was critical in the realization of this work.

I am deeply grateful to my supervisors Dr. Paul Fazio and Dr. Krishnan Gowri for their continuous assistance, constructive comments, and thoughtful recommendations. I am amazed by their patience in guiding me throughout the years, and I must thank them for spending countless hours in revising the progress of my research.

I would like to express my gratitude to Dr. Dorel Feldman whose encouragement and academic advice were inspiring me to bring this work to completion.

Most of my Artificial Intelligence knowledge comes from Dr. Rajjan Shinghal. I would like to thank him for helping me understand and master concepts that, at the very beginning, appeared to be so difficult.

Finally, I would like to acknowledge the support of my family that stood behind me over those long years, and gracefully accepted that most of my weekends belong to a work that seemed to never end.

This research was partially sponsored by FCAR and by NSERC grants held by Dr. Paul Fazio.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION AND OVERVIEW

1.1. RESEARCH MOTIVATION

Faulty designs account for a large percentage of the building-envelope failures. In their quest to eliminate design-induced failures, building professionals in growing numbers are relying on computerized tools offering some form of design assistance, from performance simulations to actual design recommendations.

Assessing the most suitable envelope for a given building design is a complex process involving the analysis of huge sets of parameters related to material and energy costs, structural and environmental requirements, durability, maintenance, and aesthetics. Moreover, the new design should comply with established construction practices, building codes, and standards.

The first step to enable computer-based designs is to identify the mathematical models that can be used to simulate the building-envelope performance for different construction alternatives. In this direction, a comprehensive simulation model is provided by the ASHRAE Standard 90.1/1989, "*Energy Efficient Design of New Buildings Except New Low-Rise Residential Buildings*". Based on extensive research in energy conservation in buildings, the Standard includes new or refined principles for building design, having the capacity to recognize advances in the performance of various components and equipment. The Standard fosters innovative energy-conserving designs by encouraging the designers to take into consideration the dynamics that exist between the many components of a building through the use of the System Performance Criteria (SPC) or the Building Energy Cost Budget Method. Using the ASHRAE Standard 90.1/1989 for efficient thermal performance of a new building means making changes

and reassessments in the approaches used in building design. In addition to thermal performance requirements, it is often necessary to accommodate owner's preferences with respect to certain aspects of the building envelope, such as exterior and interior finishes, durability, and maintenance costs.

The second step in enabling computer-based designs is to identify the procedures that link the complex design requirements that a building envelope must satisfy. Successful building designs are fueled by good knowledge and in-depth experience. No one has succeeded so far in capturing the building-envelope domain knowledge in a limited set of general rules that can be applied to any situation. Field experience through analysis of known solutions is one of the keys to designing better building envelopes.

Building-envelope practitioners are currently missing a computerized design tool that enhances the well-established domain knowledge with the expertise elicited from a collection of proven design cases. The preliminary stage of the building-envelope design process can benefit from a computer tool developed around a case-based reasoning model for recommending the most suitable construction alternative based on a library of prototypical building-envelope cases. This conclusion is supported by the following observations:

- The building-envelope design is an information-intensive process (i.e. the number of parameters and/or features to be considered is large);
- Experiential knowledge has a major impact on the building design outcome;
- Usually, the design problem is not completely described in the initial stage;
- The compliance checking is mathematically intensive and requires computer assistance (see for example, in Appendix A, the equations recommended by the

System Performance Criteria for assessing the thermal performance of the building envelope);

- The design process is based on successive trial and error iterations, with failure rates relatively high, so facilities for repair and recovery are necessary.

In summary, the building-envelope design process follows a reasoning model anchored on past experiences, and it requires computer assistance for validation and performance simulation. Therefore, there is a definite need for a computerized model including a case reasoner enhanced by a set of advanced compliance checking tools. This research aims to address that need.

1.2. OBJECTIVES

The general objective of this study is to develop the computer model of a *Case-Based Reasoning (CBR)* framework for recommending the most suitable construction alternative for a new building envelope based on a library of prototypical building-envelope cases. The specific objectives of this research are as follows:

- *To assist the preliminary building-envelope design process through a software design tool incorporating the ability to recall previous known design solutions and to adapt them to fit different sets of requirements.*
- *To develop a Case-Based Reasoning framework for selecting construction alternatives during the preliminary stage of the building-envelope design.* The methodology presented is geared toward finding the most suitable design for a new building envelope starting from a library of prototypical building-envelope cases.
- *To demonstrate several concrete solutions to some the major design roadblocks*

encountered while attempting to computerize the building-envelope design process. The major design decisions encountered while developing the case-based approach are:

- Selecting the appropriate model to represent a building-envelope case;
- Compromising the case description to avoid unnecessary encoding complexity;
- Simplifying the classification criteria to facilitate retrieval;
- Defining a set of heuristics to improve the best matching procedure;
- Inferring missing information from the available construction guidelines;
- Producing the mechanisms to transform, test, and eventually repair a building-envelope case.

- *To implement the proposed CBR framework in a software system that will validate the model, and which can be further used by building design professionals to browse and select building-envelope alternatives.* The proposed system integrates several AI techniques, such as *Artificial Neural Networks (ANN)*, CBR and *Rule-Based Inference*, and it incorporates a knowledge-acquisition facility that dynamically updates the knowledge base.

This research aims to offer a significant improvement over the existing *Knowledge-Based Expert Systems* (KBES) in terms of the level of assistance a computer tool can provide during the preliminary stage of building-envelope design. It is known that designers often learn from the experience of completing each design, thus, improving their understanding and ability to solve future design problems. The integration of knowledge-based and case-based reasoning

approaches will provide an environment in which the dynamic updating of the knowledge base as well as incremental refinements of a first, best-fit solution can be realized.

1.3. DOMINANT LITERATURE

Commonly used approaches to automate the building-envelope design process are KBES that employ a set of general rules to generate a new design (Fazio, Bedard, and Gowri, 1989, Bedard and Gowri, 1990).

CBR (that is, the technology that is used to exploit experience in problem solving) evolved from theoretical discussions (Schank, 1982, Carbonell, 1983, Kolodner, 1993) to actual implementation such as CYRUS (Kolodner, 1983), MEDIATOR (Simpson, 1985), CHEF (Hammond, 1989), PROTOS (Bareiss, 1989), CASEY (Koton, 1989), CLAVIER (Hennessy and Hinkle, 1992), etc.

Engineering design was an early target for CBR (KRITIK – Goel, 1989). However, in building design, CBR had a lower impact, and was limited to simple tasks such as browsing alternatives (ARCHIE – Domeshek and Kolodner, 1992) rather than suggesting solutions.

1.4. LIMITATIONS OF PREVIOUS KBES MODELS

KBES encode general concepts and domain knowledge in a set of rules that are furthered used to infer design recommendations.

CBR systems favor experiential knowledge encapsulated in previous design cases to derive recommendations for a new design through a process that has at least two steps: one, matching, and two, adaptation.

Though KBES proved to address many issues of representing symbolic information, previously developed systems (Fazio, Bedard and Gowri, 1989) have serious limitations due to the static nature of their knowledge base. It is known that designers often learn from the experience of completing each design and, thus, improve their understanding and ability to solve future design problems.

There is no fully qualified CBR approach to building-envelope design. All previous attempts were either limited in scope (i.e. confined to a single step), or could not offer solutions when confronted with common problems such as incomplete description of the design problem.

1.5. PROPOSED APPROACH

This research proposes to combine the benefits of both techniques (i.e. KBES and CBR) in a unified methodology:

- Employ a CBR approach to account for experience by identifying in a database of previous designs the closest match for the current set of requirements.
- Employ an ANN-based filtering mechanism to improve the quality of CBR while preserving the economy of knowledge representation.
- Employ a KBES approach to infer missing information, to adapt and eventually repair the retrieved case.
- Use the well-established domain knowledge to validate the envelope design and to predict its performance.

Historically, the proposed approach evolved from two sources:

a) The decomposition model for a building-envelope case developed by Gowri, Fazio, and Bedard at the Centre for Building Studies (1988-1990).

b) The performance simulation techniques recommended by the System Performance Criteria of ASHRAE Standard 90.1/1989.

The main advantage of the new model consists in integrating the general building-envelope concepts with experiential knowledge. However, this approach has not been validated yet by practitioners and was not fully tested under real-life conditions.

1.6. METHODOLOGY

This research combines three aspects as follows:

- Analytical work to develop a CBR model for building-envelope design.
- Data acquisition work to collect a set of prototypical building-envelope cases, and to encode them according to the above model.
- Implementation effort to transfer the model into a software program for building-envelope design (and thus to validate the concept).

A computerized model of the building-envelope design process is developed within a CBR framework. The model includes:

- The minimal set of features and properties identified to capture the essential characteristics of a building-envelope entity.
- The encapsulation model required for encoding a building-envelope case along the above-identified features.
- The modality to store and classify building-envelope cases accounting for experience.
- The mechanism to retrieve and adapt an existing building-envelope case when provided with a new design context.

With respect to data acquisition work, a comprehensive set of building-envelope cases was assembled through discussions with practitioners, and from the available literature. The cases are selected to cover most of the common design situations encountered for buildings of different types and sizes.

The model validation is attempted by implementing it in a computer program for the preliminary building-envelope design. The program uses a database of known building-envelope cases to suggest design recommendations when fed a set of requirements for a new building. The developed model demonstrates how experiential knowledge can be incorporated to enhance a computer-generated envelope design, and how the integration of the KBES and CBR approaches can be realized.

1.7. ANTICIPATED VALUE OF THE WORK

This research presented herein has lead to:

- A CBR framework that can be used as a guideline for implementing a computerized tool for building-envelope design.
- A methodology for mixing several AI technologies (ANN, CBR, KBES) with compliance-checking procedures to enhance the building-design outcome.
- A database of successful building-envelope cases that can be further augmented and refined.
- A software program able to assist practitioners during the preliminary stage of the building-envelope design process in browsing and evaluating different design alternatives.

2. PROBLEM DEFINITION AND DOMAIN KNOWLEDGE

The primary goal of the building envelope is to protect and maintain the comfort conditions of the indoor environment from the rapidly fluctuating outdoors climatic conditions. In achieving these objectives, the building envelope must fulfill the following functions:

- control of heat flow,
- control of water vapor flow,
- control of rain penetration,
- controlled admittance of natural light and solar radiation,
- control of air movement through wall,
- stability against wind pressure and the regulation of differential air pressures,
- protection against fire,
- control of differential thermal expansion,
- control of vibrations and seismic stress,
- durability combined with low maintenance,
- control of outdoor noise, and
- economy.

In designing an envelope to conform to these requirements, it is essential to examine each condition in detail to understand the principles and mechanisms involved. Though a wide variety of design guidelines, research reports, and trade literature is available, there does not yet exist a methodology by which this information can be used in comprehensive manner to assist designers in evaluating building-envelope performance. In the present study, a comprehensive set of

literature sources has been identified to develop a knowledge base, which can be implemented in a computerized system.

2.1. BUILDING ENVELOPE DESIGN FOR ENERGY EFFICIENCY

The quest for energy conservation led to the increase of building insulation. Reducing the heat flow through the assembly by increasing the thermal resistance of the envelope raises questions about how much insulation is really necessary and cost-effective. According to the ASHRAE Standard 90.1/1989, the optimal values for all building-envelope thermal parameters are to be calculated based only on the general building characteristics (such as type of building, geometry, orientation, and so forth) and the average climatic data of the selected location (such as heating and cooling degree-days, average incident solar radiation, and so on).

Using the average weather data of the building site, and other additional information concerning the type, geometry, and orientation of the building, one may generate, based on ASHRAE Standard 90.1/1989, a theoretical model of the building, called the *prototype building*, and use it as the reference for achieving energy efficiency. The new building design will be considered to meet the Standard requirements if, and only if, its thermal performance will be at least as good as the thermal performance of the associated prototype building; that is, the following relation holds:

$$H_{BLDG} + C_{BLDG} \leq H_{PROTOTYPE} + C_{PROTOTYPE}$$

where H_{BLDG} and C_{BLDG}, are respectively the wall heating and cooling load values (i.e. cumulative annual heating and cooling energy flux) for the new building, and $H_{PROTOTYPE}$, and $C_{PROTOTYPE}$ are respectively the wall heating and cooling compliance values (i.e. cumulative

annual heating and cooling energy flux) for the prototype building. More details about the calculations required by the ASHRAE Standard 90.1/1989 are to be found in Appendix A.

Design requirements are further complicated by the existence of local construction codes, such as the Quebec Law for energy efficiency. Hence, any new building design should not only respect all regulations found in local building codes but also pass the energy efficiency tests. Appendix B provides details on how ASHRAE Standard 90.1/1989 could be combined with the Quebec Law in generating the theoretical envelope for a prototype building in Montreal.

The prototype building provides sufficient data for employing simulation tools such as BESA, DOE-2, BLAST, and so forth, to assess the impact of varying the insulation value on the overall thermal performance of the building. For example, Appendix C shows how the total energy consumption of a sample building is affected by modifying the thermal insulation of the walls.

The total energy consumption of the building decreases with the increase of the overall thermal resistance of the walls as shown in Figure 2-1.

Using regression analysis on the data gathered by studying the energy consumption of the building with respect to the overall thermal resistance of the walls, a non-linear relationship can be obtained:

$$Energy\ Consumption = E_0 \times (1 + a \times e^{-b \times R}) \quad \frac{kWh}{m^2}$$

where R is the thermal resistance of the wall, E_0 is the minimum value expected for energy consumption, and a and b are two constants in the 0 ... 1 range.

In the example of Appendix C, an increase of the total thermal resistance of the wall from 0.5 to 5 times the recommended value of ASHRAE Standard 90.1/1989 (that is, 2.5 m^2C/W) reduces the overall energy consumption by about 13%. A simple explanation is that increasing

thermal resistance improves heating performance, but deteriorates cooling performance. The conclusion emerging from the above analysis is that the impact of increasing the thermal resistance of the wall over and above the ASHRAE recommended value on the energy performance of the building is limited (in general, to less than 20%).

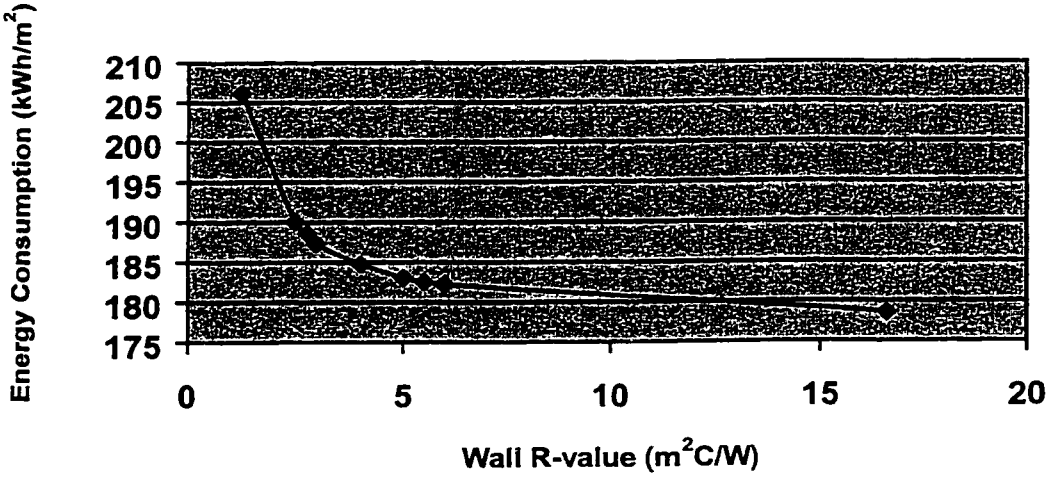


Figure 2-1. Energy Consumption vs. Wall R-value

Therefore, even if energy efficiency of the building is a desirable trait, increasing it through improvements in the thermal performance of exterior walls should be carefully considered. Analysis of energy savings should be accompanied by analysis of material costs implied by any improvements. The question to answer is how much money should be invested towards achieving the energy efficiency of the building through envelope improvements. In this direction, one recommended method is the life cycle cost analysis of the relationship between the insulation cost and the energy saved (Masonry Council of Canada, 1982). Appendix D contains a

sample of such life cycle cost analysis showing that the R-value should not be increased more than 36% over the ASHRAE recommended value. The 0.36 value will be later used as a heuristic threshold to limit the wall R-value modification range when attempting to repair a building envelope that fails the energy-performance requirements.

By combining the recommendations of ASHRAE Standard 90.1/1989 with other forms of economic analysis, an optimal thermal resistance of the wall can be obtained, a value that takes into account the energy savings over the entire life of the building (usually estimated to be about 20 years).

2.2. PRACTICAL CONSIDERATIONS

Traditionally, in cold climates, the air-vapor barrier has been placed on the warmer side of any insulation. This is intended to prevent moisture from entering the wall where cooling can cause condensation. However, in energy-efficient houses, it is desirable to place the air-vapor barrier between layers of insulation. This protects the barrier from damage during construction. Instead of being penetrated by plumbing and electrical outlets, or damaged during drywall installation (particularly common for walls), the barrier can remain intact. Hence, Canada Mortgage and Housing Corporation (CMHC) suggested the arrangement shown in Figure 2-2 in 1982 as a safer alternative toward lowering the maintenance costs and improving the overall performances of the wall. The installation costs will be higher, however, due to the increase in complexity. In our computerized approach to building-envelope design, embedding vapor barrier within layers of insulation will be the preferred layout whenever the insulation thickness exceeds the threshold of 76 mm (3 inches).

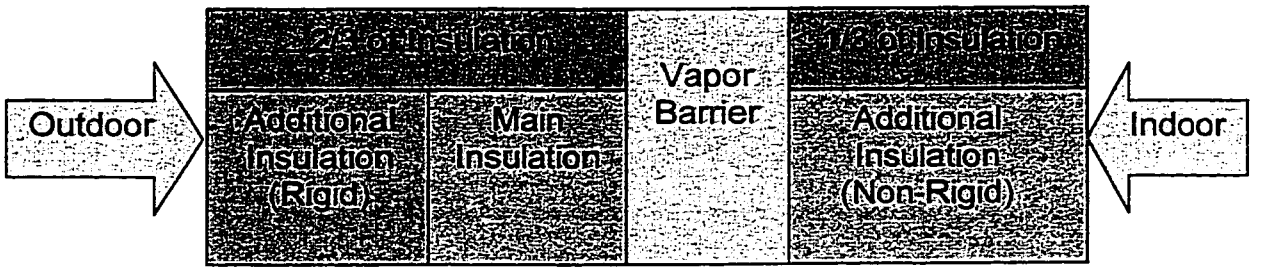


Figure 2-2. Insulation Placement

Thermal and moisture performances of any buildings, whether a simple dwelling or a complex multi-story building, are heavily affected by poorly installed insulation and by structural penetrations providing thermal bridges for heat flow. A series of tests performed by CMHC has shown the impact of insulation placement on the temperature profiles for Veneer Wall Systems, and, consequently, on the resulting dew point locations (Johnson, 1991). Of the five tested assemblies, only the one having the insulation located on the exterior of the sheathing had all temperatures above the dew point.

It was not until the 1980s that the thermal bridge occurring at each stud location became a serious concern. Most, if not all, recent structures have a layer of insulation applied to the face of the sheathing in order to reduce the effect of that thermal bridge. This provides a number of advantages, such as:

- The air-vapor barrier is in the same location, on the warm side of the insulation.
- There is virtually no thermal bridging at stud location or slab edge.

- By being applied on the exterior face, there are fewer obstructions to maintaining a complete seal.
- The interior face of the wall can be penetrated at will for such things as convenience outlets, without fear of damaging the building envelope.

Following this idea, in an optional reasoning path of our computerized approach to building-envelope design, rather than insulating the space between studs, the entire insulation is placed outside the studs, thus avoiding the problems caused by thermal bridges. There are, however, some disadvantages in this approach, such as:

- The assembly is quite thick although the stud size may be reduced as it is now dependent on wind load and not on insulation thickness.
- The cladding must be supported at some distance from the main structural frame.
- If problems occur, the assembly can only be accessed by removing the outer cladding.

2.3. MOISTURE FLOW THROUGH THE BUILDING ENVELOPE

The recent requirements for increased thermal efficiency have introduced some new problems to be considered. Disregarding comfort and energy costs, older, poorly insulated assemblies had few problems since the entire wall was constantly kept warm and air movement was relatively unimpeded. If moisture did gain entry to the assembly, either by rain penetration or condensation, it would not freeze since the air movement provided a good drying mechanism. In new assemblies with high thermal resistance, some components of the assembly will be below the dew point, and, in some circumstances, below the freezing point. Walls have low drying potential, and if moisture is allowed to accumulate in such an assembly, whether by vapor

transport, air leakage, or rain penetration, substantial damage to the components is bound to occur. Therefore, to avoid damage in well-insulated assemblies, control of vapor transmission, air leakage, and rain penetration must be a priority.

The two possible mechanisms of vapor flow through a wall are:

(a) vapor diffusion (under vapor pressure gradient);

(b) vapor transport (under air pressure gradient, i.e. air flow).

The amount of air moving through any opening in the envelope is directly proportional to the pressure difference across that opening. Pressure differences can arise from wind, stack effects, and mechanical ventilation systems:

- Wind will cause an overpressure on the windward face of a building and a suction pressure on the leeward face. Although the effects of wind are well understood, little, if anything, can be done to control these natural effects of building pressurization.
- In cold seasons, the stack effect in buildings is the same as the stack effect in chimneys; i.e. the hot air rises. The result is that the upper portion of the building becomes positively pressurized and the lower portion negatively pressurized. While the stack effect might not be eliminated, the airflow through the openings can be reduced through various methods such as increasing air tightness of exterior enclosures and interior separations, and adjusting air-handling systems to provide an imbalance of supply or exhaust.
- Building pressurization is affected if the ventilation system exhausts more or less air than is taken in. Management of pressurization is dependent upon the sophistication of the system and its controls.

It is clear that pressure differences across the envelope cannot be avoided. Airflow and vapor flow by themselves cause little concern (if the energy loss aspect is overlooked). It is only with the introduction of heat flow via a temperature gradient that it becomes possible for the vapor to condense into water or frost. Water within walls is a prime factor in the deterioration of wall systems.

2.3.1. Surface condensation

Thirty years ago, the major concern was to ensure that visible or surface condensation did not occur on interior wall surfaces (Hutcheon and Handegord, 1989). Glazing was designed to provide sufficient thermal resistance to maintain the inner surface above the temperature that would cause condensation to occur. That is, temperatures were kept above the dew point. Air spaces in multiple glazing were sealed to minimize moisture content. Building codes specified that sufficient insulation must be provided to maintain the interior wall surfaces above the dew point. Thus, the first problem that was addressed and essentially solved was visible or interior surface condensation.

2.3.2. Concealed condensation

Unfortunately, the majority of severe serviceability problems result from concealed condensation, which is the condensation of water vapor occurring within the wall system. Central to the issue of concealed condensation are the mechanisms of vapor transfer, that is, vapor diffusion and air leakage.

Air leakage is the uncontrolled movement of air through the walls, both into a building (infiltration) and out of it (ex-filtration). Condensation resulting from ex-filtration during cold

weather can cause rapid deterioration of the building wall elements. In a cold climate, the infiltration of cold dry air is less damaging to the wall assembly than ex-filtration of warm moist air. Therefore, it would be preferable to maintain a slightly negative building pressure. If the envelope is poorly sealed against air leakage, even the most sophisticated ventilation system would not be sufficient to maintain a negative pressure. This leads to the conclusion that uncontrolled airflow across the envelope must be reduced to an absolute minimum, especially in tall structures where wind and stack effects have the most impact.

As a result of growing concern regarding the effect of uncontrolled air leakage in buildings, an "effective" air barrier within the building envelope is required in all new buildings.

2.3.3. Air barriers

To effectively accomplish the control of air movement, the air barrier should be linked continuously within the building envelope. Continuous air barriers at corners of buildings are essential to resist greatly increased wind pressures. This, however, is not always achieved in reality. Based on recently reported research (Johnson, 1991), the air barrier assembly should be considered as a separate element within the building envelope and designed to meet the following criteria:

- Support and transfer the air pressure to the structure of the building.
- Materials used as air barrier should be as "air impermeable" as possible (e.g. gypsum board, concrete, reinforced sheet membrane, and so on) but not necessarily vapor impermeable.

- Though complete air tightness is not possible, the air leakage should be limited by avoiding any visible openings, cracks, fissures, or holes on the surface of the air barrier.
- The air barrier assembly should be rigid so that, when it is subjected to an air pressure difference, it can resist the structural forces. The joints between different elements of the air barrier, however, may be flexible.
- The air barrier should perform for the life of the building, or provision should be made to allow the maintenance of the assembly.

It must be remembered that for an air barrier to be "effective" it must be continuous. This means that the barrier is not simply a membrane but a system of components that provides resistance to air movement. Those components might include the roof membrane, wall sheathing, door and window frames, and the foundation walls. Not only the air permeance of components is important, but also the seal of the connections between these components requires careful attention. The connections may be caulked or have a flexible membrane or gasket applied. The nature of the connection is dependant upon whether it is exposed to weather, extreme temperature differences, structural movement, and whether the joints are accessible for maintenance.

2.3.4. Vapor retarders

Vapor diffusion is the result of vapor pressure difference that causes water vapor molecules to migrate through most materials. The amount of water vapor passing through the building wall will depend upon the permeance of the wall assembly and the vapor pressure difference between the inside and outside air.

All building materials offer some resistance to water vapor diffusion; some materials, called vapor retarders (or vapor barrier), offer more resistance than others and are used to reduce the diffusion of water vapor. The most common available vapor retarders include polyethylene, asphalt coated felts, asphalt-laminated papers, metal foils (copper, aluminum), paints, and coatings.

The vapor barrier should control the movement of air through the wall such that no warm moist air will pass far enough into the wall system to sufficiently cool down and condense. This implies that the vapor barrier must be on the warm side of the insulation.

In contrast with air barriers, the continuity of vapor retarders is not critical. Typically, material overlaps at joints are sufficient to reduce vapor diffusion and, hence, sealing of the overlaps is not required.

2.3.5. Potential for moisture-related problems

In many buildings, air leakage through the building envelope has been identified as the number one problem. The volume of air involved in air leakage may not be significant in terms of heating and ventilating the building, but the amount of moisture carried out by air from inside a humidified building can be large enough to cause serious performance problems.

Although most new buildings have more wall and roof insulation than older buildings, the cladding, wall, and envelope damage due to air leakage through the building envelope has increased significantly. With more insulation in walls, cladding is subject to wider extremes of temperatures; the outer wall elements are maintained at or near the outside temperature. The colder cladding triggers more condensation with subsequent water damage. Under freezing conditions, the water migration can grow into ice lenses, causing expansion and displacement of

exterior veneers. Humid air flowing through a building envelope in an uncontrolled fashion often results in the above-mentioned problems, thereby causing damage to the cladding and exterior walls of new buildings. The most common moisture-related problems in wall systems include the following:

- deterioration of exterior sheathing;
- deterioration of insulation;
- freeze-thaw damage of exterior facing;
- excessive efflorescence;
- mortar-joint spalling;
- masonry discoloration;
- corrosion of metal components.

2.3.6. Water penetration

Rain penetration through exterior walls occurs all too frequently despite advances in building technology. The combination of water and the wall surface fissures, which allow its passage, propels the water inward producing rain penetration. Rain penetration cannot occur without all these conditions being present.

During heavy or persistent rain, a film of water will form on the exposed exterior wall surface. Air current, surface texture, and gravity govern the flow of this water, generally in a downward diagonal flow that collects in protected vertical joints. Structural and thermal movement in the exterior wall surface, when not considered in design, may produce cracks, fissured joints and weakly bonded interfaces, which, combined with porous materials, provide

many opportunities for water penetration. Impermeable facings have furthered the development of impervious wall surfaces but site application techniques and solar radiation on exposed sealant make it impossible to maintain a wall permanently free from cracks and openings. The exterior wall should be designed using proven rain-screen design principles where water entering the wall will be drained using appropriate flashing details.

Most cavity walls adopt the open rain-screen principle in which the wall consists of three distinct parts:

(a) a vented outer layer or wythe,

(b) and an insulated inner airtight wall, and

(c) an air chamber (air cavity) separating the two layers.

The air pressure in the air chamber space must be equal to the pressure on the exterior wall. This is accomplished by providing combination of weep holes and drainage vents in the outer wall. Flashing is located along the entire wall at the bottom of the weep holes to permit any water that has entered the rain-screen to escape to the atmosphere.

Openings such as windows, doors, and grilles in multi-layer walls must be sealed to the air barrier inner walls with projection or bulkheads connecting with the outer rain-screen. The air barrier must prevent major air leakage and resist wind loads on the building. The air space should be closed at corners to prevent air from going around the corners and feeding possible high suction on the adjacent wall face.

A complete rain-screen design approach can result in the following advantages:

- permits rapid drying of cladding material,
- permits cladding movement and crack control,
- permits better positioning of insulation minimizing condensation risk within the wall,

- structural elements are maintained at a more uniform temperature, with minimized expansion-contraction range.

2.4. AUTOMATING THE BUILDING ENVELOPE DESIGN

The complexity of the building envelope design process is due to several factors such as:

- the large number of parameters to be considered (parameters related to building materials, energy efficiency, structural and environmental requirements, durability, maintenance, aesthetics and costs),
- the overlapping and competing nature of compliance guidelines (construction practices, building codes, standards, etc.),
- the lack of methodology for processing design information in a comprehensive manner,
- the need to combine well-established domain knowledge with experiential knowledge (that is, expertise elicited from proven design cases).

To successfully handle that complexity and to avoid design-induced failures, the building professionals need computer assistance.

Ideally, any computerized tool involved in engineering design aims to reach a level of human expert competence. Human problem solving and learning are processes that involve the representation of several types of knowledge, and the combination of several reasoning methods. In the case of building-envelope design, a high level of expertise cannot be achieved without acquiring good quality knowledge through experience. The better the quality of previous design experience, the higher the likelihood of future fault-free designs. The problem-solving paradigm where reasoning is supported by reusing past cases is called *Case-Based Reasoning (CBR)*, and it

emerged as an AI area of interest following research in cognitive psychology. Next section presents an overview of the CBR methodology, that is the approach adopted by this research to automate the designing of building envelopes.

If cognitive plausibility is a guiding principle, an architecture for knowledge-based systems where the reuse of cases is at the center, should also incorporate other and more general type knowledge in one form or another. This is an issue of current concern in CBR research. Following the general trends of knowledge-based systems (shown in Figure 2-3), this thesis attempts to address that concern by balancing the CBR methodology with other AI techniques such as ANN and rule-based inference.

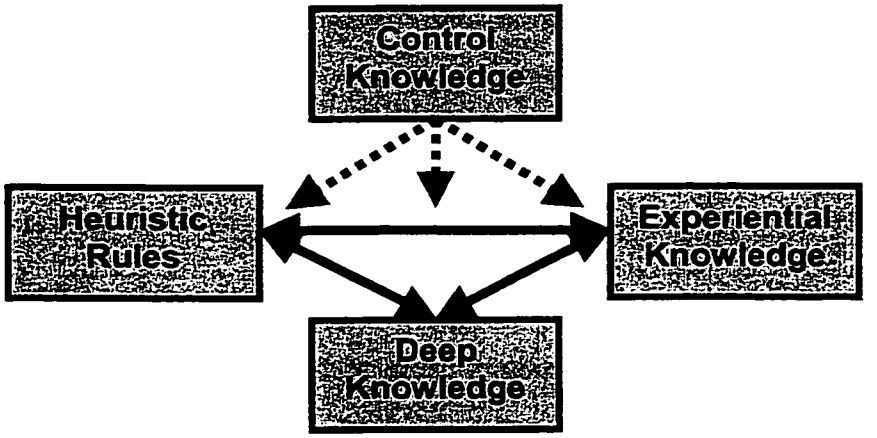


Figure 2-3. Trends in Knowledge-Based Systems

Some of the major reasons for applying a CBR approach to an engineering problem (such as the designing of building envelopes) are the following:

- Being reminded of previous experiences is a powerful way to solve new problems.
- Learning by sustained adaptation of past problem-solving solutions is valuable.
- Learning of generalizations is more difficult than learning of specialized cases.

3. CASE-BASED REASONING PARADIGM

Case-Based Reasoning (CBR) is a problem-solving paradigm where a new problem is solved by remembering a previous similar situation and by reusing the information and knowledge of that situation. In CBR terminology, a *case* usually denotes a specific problem situation including its interpretation, solution, and possible annotations. A case is a previously experienced situation, which has been captured and learned in such a way that it can be reused in solving future problems.

CBR emerged initially as an AI paradigm following research in Cognitive Psychology, but it was further adopted as a technology for various industrial and business applications. CBR became an alternative not only to rule-based expert systems, but also to analysis tools based on statistics and information retrieval. CBR technology is suitable to domains that exhibit characteristics such as (Althoff, Aurial, and al, 1995):

- Domain has weak or unknown causal model;
- Contradictory rules apply in different situations;
- Domain formalization requires too many rules;
- There exists records of previously solved problems, and historical cases are viewed as an asset that ought to be preserved;
- Specialists talk about their domain by giving examples;
- Experience is at least as valuable as textbook knowledge.

All the above criteria apply to the building-envelope design process.

3.1. CBR PREMISES

CBR suggests a model of reasoning that incorporates problem solving, understanding, and learning and integrates them with memory processes (Kolodner, 1993). CBR emphasizes the use of instances, called cases, over abstract domain knowledge as the starting point for reasoning. Simpson (1985) showed that the CBR model is based on the following five fundamental premises:

(a) *Reference to previous cases is possible through a dynamic memory accounting for experience.*

Thus, remembering a case to use in later problem solving (and integrating the case with what is already known) is a mandatory learning step. However, the library of cases should be kept at a reasonable size by limiting it to the so-called "paradigm" cases, which are past experiences linked to relevant and unique information regarding either success or failure.

(b) *Capabilities to recognize similarities between cases are available.*

Because descriptions of the problem are often incomplete, a good understanding or interpretation of the problem is a necessary prerequisite for reasoning. A case-based reasoner cannot recall a relevant case unless it understands the new situation. Moreover, as problem-solving progresses, a reasoner may gain a better understanding of a situation, allowing more relevant cases to be recalled. This suggests that understanding or interpreting the new situation is a necessary part of the reasoning cycle. In conclusion, an efficient CBR mechanism should:

- recall only those cases that are potentially applicable, and retrieve the same cases no matter how many cases were screened;

- while searching for similar cases, rather than looking for individual goals and then merging the results, select only those cases that satisfy many of the required goals at once.

(c) *Mechanisms to transfer information between cases are at hand.*

Since no existing case will ever be exactly the same as the new one, it is usually necessary to adapt an old solution to fit a new situation. Adaptation compensates for the differences between an old situation and a new one. The reasoner must be able to transform the "best match" case according to current requirements using:

- a knowledge base of modification rules, and
- domain knowledge about the design constraints.

(d) *Learning occurs as a natural consequence of reasoning.*

If a novel case is derived following a reasoning process, then the new case is saved in the dynamic library of cases, and indexed according to its salient features, such that a later retrieval will benefit from it. So, rather than discarding the cases it builds, a case-based reasoner has to save them in memory for later use. If problems are encountered while applying an old case to a new situation, the reasoner must be able to recover from reasoning errors. In the process of recovery, the case-based reasoner should treat these errors as opportunities to learn more about its domain and the problems that arise in it.

(e) *Feedback and analysis are part of the complete reasoning/learning cycle.*

Without the evaluation process based on feedback, learning could not happen, and references to previous experiences during reasoning would be unreliable. Complete and/or partial design solutions are checked through simulation to reveal

inconsistencies. Testing is applied recursively till the design is complete. Errors may arise from:

- wrong selection of similar cases;
- unacceptable adaptation;
- initial goals are either contradictory or too restrictive to be achieved.

The above premises suggest that the reasoning quality in a CBR system depends on several factors such as:

- the amount and diversity of cases supporting the reasoning system;
- the ability of the system to understand a new situation, relative to those old cases;
- the capability to adapt old cases to new contexts;
- the level of evaluation and the capacity for repairing selected cases;
- the ability to filter new experiences and to integrate the significant ones within the dynamic memory of cases.

Key issues when building a CBR system are:

- representing the case so as to capture its true meaning;
- indexing cases to retrieve them quickly;
- assessing the similarity between a current case and the retrieved ones;
- adapting a solution that worked in the past to the new problem;
- checking the validity and conformity of the new solution.

In conclusion, the central problem of case-based reasoning is what exactly is meant by a "case", and how to represent it. The main processes of case-based reasoning are *case retrieval*, including input pattern matching, and *case storage*, linked to memory update and case evaluation.

3.2. DEFINING A CASE MEMORY

It is often said (Riesebeck and Schank, 1989) that a case-based reasoning system can be only as good as its memory of cases. The case memory includes not only the library of cases but also the set of access procedures defined to store and/or retrieve information from that repository of cases. Therefore, one of the important problems to be solved in implementing a CBR system is the case representation, which will raise several major design issues (Hammond, 1989):

- What knowledge should a case encode?
- What are the relevant features of a case, or in other words, why is a particular case a meaningful one?
- How should the cases be indexed and linked within the library of cases?
- What methodologies are used to access the cases, retrieve their information, and, eventually transform them to fit a new context?
- Why is a case "worth remembering" and how will be the library of cases dynamically updated?

In a general definition, a *case* is a contextualized piece of knowledge embedding a domain specific lesson about how to achieve a given goal (Kolodner, 1993). For instance, in the building-envelope design domain, a case could include not only the full description of building-envelope elements but also some detailed design knowledge about what strategies should be applied to achieve energy efficiency, or how general guidelines derived from well-established building-construction practice should be considered in the given context.

A case can be regarded as having three major features:

- its *description*;
- its associated solution (also called as its *diagnosis*, its *class*, or its *target*);
- the *justification* of its solution.

The description includes all the relevant attributes (or features) of the case. Those case attributes with a significant impact on the case behavior and performance are usually called *salient features*.

One has to distinguish between cases that augment the system's knowledge and are worth being remembered (that is, stored in the library of cases) and cases that duplicate existing information, without making a significant difference to what is already known. Therefore, a design instance that is to be kept in the case library, usually called *prototypical* or *paradigm case*, has several characteristics, such as:

- It represents a specific piece of knowledge tied to a context, recording that knowledge at operational level rather than at conceptual level.
- It records an experience that is different from expected, either by how it achieves a goal or by how it avoids a failure.

It is important to note that the value of a case is directly linked to the appropriate circumstances in which the case was defined. In other words, a case is indexed by its functionality context that will govern the condition under which the case may be retrieved. For example, a high-rise building design will be a case so indexed that its retrieval for the design of a one-story family house will be, at best, unlikely if not impossible.

Therefore, to define and organize a library of cases, one should address the following CBR issues:

a) Establish a selection criteria

Accumulating a very large number of cases is restricted by available storage and increases the difficulty of retrieval. So, cases worth remembering should be properly identified.

b) Select case's indexes

Indexes are those "salient" features of a case which specify the appropriate circumstances to retrieve that case (that is, the context in which case information provides some meaningful solutions).

c) Enforce economy of representation

Store only useful information from a given case.

The organization of the case memory can follow one the following models:

- *flat representation*, where each case is described by the same attributes;
- *structured representation*, where each case is indexed in a hierarchy tree, the arguments used to described being defined by the indexing tree;
- *hybrid representation* (a mixture of flat and structured representation).

3.3. FEATURE SELECTION

At the core of case representation lies the problem of feature extraction, since features are the attributes (or characteristics) that distinguish a given case from the others. Those features that encapsulate the case functionality and delineate its most important characteristics are usually called *salient features*. The salient features used to index the case in the memory are called *indexes*. There are two major approaches to features selection:

(a) automated feature extraction (i.e. features are selected by a computer), and

(b) manual feature extraction (i.e. features are selected by a domain expert).

In the first approach, the design domain (in our case, building envelope) is described along some important parameters, and each case is represented according to those selected parameters. The case, so encoded, is then submitted to a computerized classifier (e.g. an artificial neural network) that will extract the appropriate feature vector. In the second approach, the library of cases will be analyzed by a domain expert, who will decide the case representation by indicating which are the relevant features.

Neither solution is perfect; the machine may focus on some meaningless features, while the human expert may overlook some of the not so obvious patterns. This research recommends employing both approaches, with the human expert making an initial, coarse, selection of features, while the machine will be further used to fine tune that set of features. In essence, initial selections made by a computer are avoided, since the human experts have a better understanding of their given design domain, and the potential bias of the human experts is eliminated by using the machine to map the selected features into a vector that will finally encode the case. Given a library of cases, the following method is proposed for features extraction:

(a) Let a human expert select an initial set of features.

(b) Represent all cases along those features.

(c) Associate a conceptual hierarchy tree to be used as criteria for generalization.

(d) Let the computer browse the stored cases and explore the generalization options until the optimal set of features is selected; in other words, let the machine filter the cases based on the conceptual hierarchy tree provided.

3.4. MAJOR STEPS IN CASE-BASED REASONING

Starting with case retrieval as primary process and ending with memory update, the CBR flow has the following four basic steps:

(a) Access and Retrieval;

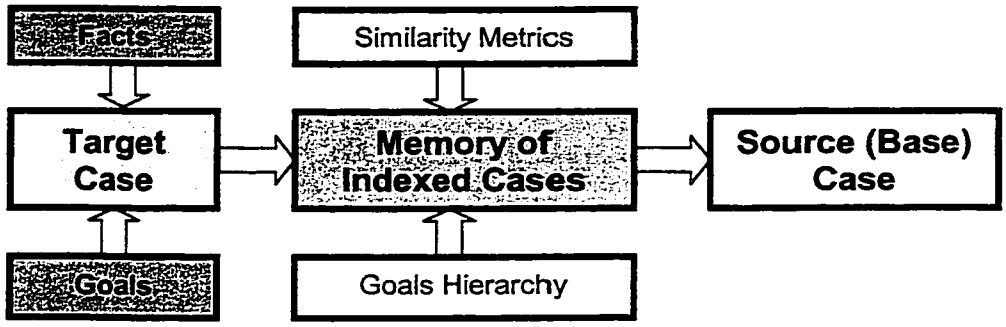
(b) Mapping (or Transformation);

(c) Evaluation and Repair;

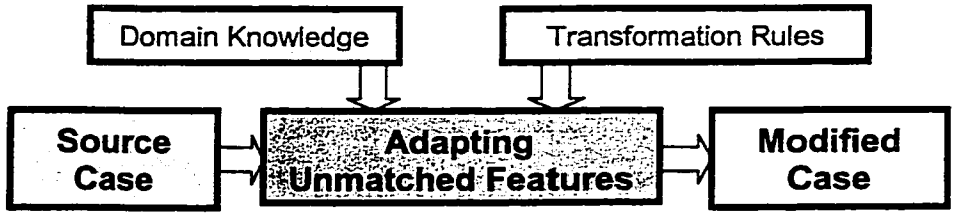
(d) Learning and Updating.

Hammond presented a reasoning flow (Hammond, 1989) that starts with a TARGET case (representing the problem to solve), continues with the SOURCE case (accounting for best match found in the case library), produces a MODIFIED case (obtained after applying the modification rules to the SOURCE case), and ends with the SOLUTION case (the MODIFIED case after passing the evaluation steps, including repairing, if necessary). Figure 3-1 details the reasoning model proposed by Hammond.

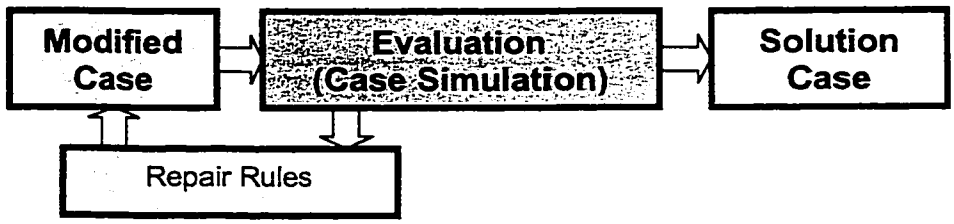
1. **ACCESS & RETRIEVAL**



2. **MAPPING**



3. **EVALUATION & REPAIR**



4. **LEARNING & UPDATING**

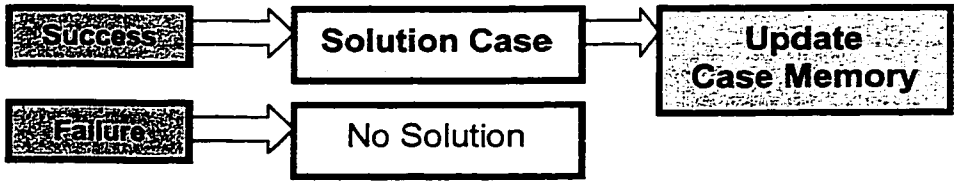


Figure 3-1. Basic Steps in Case-Based Reasoning
(From K. Hammond, "*Cased-Based Planning*", 1989)

3.4.1. Access and Retrieval

In general, it consists of two sub-steps:

(a) Recall previous cases.

A target case is assembled from the input data and is used as criteria for accessing the case library. Retrieval of similar cases from the case library is achieved by using features of the new case as search criteria. Cases that could trigger the retrieval mechanism are those that have the potential of being adapted to the new situation.

(b) Select the best case or subset of cases.

If sub-step (a) collects all cases that may be suitable to the input requirements, sub-step (b) narrows down that list to the most likely choices (a "best-match" case or, at most, a couple of cases called "best subset" of cases).

For the retrieval to happen, the computer needs to be provided with a similarity metric and a goal hierarchy, the first parameter (or function) gives the computer a means of recognizing if a case is applicable to the new situation, while the second parameter allows for weighting the usefulness of the retrieved case (or cases).

Another issue is the selection of a retrieval algorithm capable of searching a massive library of cases without slowing down appreciably the retrieval process as new cases are added to the memory. This research proposes to employ here an *Artificial Neural Network* (ANN) classifier for fast and reliable filtering of cases.

3.4.2. Mapping

The retrieved case, selected as the starting point for the reasoning process, should be so transformed as to match as many—if not all—input requirements as possible. That process, also called case adaptation or case transformation, should be performed along similar features, which are features that could be swapped without changing the functionality or the main characteristics of the case.

To map unmatched features into the retrieved case, the computer should rely on some mechanism enabling the transfer of information, consisting of adaptation rules and validated by the existing domain knowledge. For instance, the computer should know that a curtain wall is not to be transformed into a masonry wall, and, in general, that the structural layer of an envelope component can be mapped only under some particular circumstances.

The reliability and quality of adaptation rules are essential for the success of the whole reasoning process. A bad case transformation will have a negative impact on further steps, increasing the chances of failure. At this stage, it is appropriate to raise the issue of combining case-based reasoning with other reasoning models such as rule-based reasoning. Instead of looking at the two models as competitive alternatives, this study subscribes to the opinion of those authors who have seen in case-based reasoning a way to enhance rather than replace rule-based expert systems. The adaptation step could be the intersection point of the two models, with case-based reasoning submitting the SOURCE case to a rule-based system that will transform it into a MODIFIED case required by the next reasoning steps.

3.4.3. Evaluation and Repair

Usually, a solution cannot be validated until it is tried out in the "real" world; however, when the knowledge exists to simulate the validation process, we may be able to reveal inconsistencies through computerized methods. For example, it can be assessed if a given envelope component will have moisture-induced problems by simulating its dynamic behavior at different humidity levels that could occur at the given building location. Hence, performance evaluation, either through simulation or using real world tests, is a mandatory validation step for providing the CBR system with the feedback required to trigger either a repairing procedure or a learning phase.

3.4.4. Learning and Updating

A case-based reasoner has to acquire knowledge through the reasoning process, and, eventually, save it in memory. In other words, if the case built was marked either as an unique instance, or an improved one, it will become a case "worth remembering" and, accordingly, saved in the case library. The most important part of case library update is choosing the ways to "index" the new case so that during later reasoning it will be recalled when it can be most helpful.

Another aspect of learning in a CBR system is that knowledge acquired through evaluation feedback might be eventually generalized to derive empirical knowledge in a given domain.

3.5. CBR VERSUS OTHER REASONING METHODS

It is obvious that case-based reasoning is recommended for those reasoning processes that require experiential knowledge (such as the building-envelope design process). It is very useful

where there is a lot of information available but not systematically organized into hierarchies of concepts and rules. CBR is also useful when domain knowledge is incomplete or sparse as opposed to more traditional AI systems, which use certainty factors to account for inexact reasoning. It is well known that those methods based on certainty factors are computationally expensive and not intuitively plausible. CBR offers an alternative method for dealing with incomplete information, a method where missing knowledge is derived from previous experience embedded in the case description.

From the above discussion, it can be concluded that some of the benefits of using CBR technology over other AI methods are as follows:

- discovering knowledge in data;
- delivering consistent and valid solutions even when the target domain and its constraints are not fully understood by the reasoner;
- solutions could be obtained relatively fast compared with other AI methods where the problem is solved by starting from scratch;
- preserving the know-how of the most talented domain experts by capturing their experience and transferring that experience to the novice;
- providing means for dynamic evaluation and improvement of the reasoning outcome;
- helping the reasoner to focus on the important parts of the problem to solve, by pointing out what the important features of the problem are.
- building a domain memory by sharing individual experience; previous experiences augment the power of the reasoning process by avoiding past mistakes and by retrieving already found solutions.

Case-based reasoning has its share of disadvantages, such as:

- Solving problems exclusively based on previous solutions could fail in novel situations (very much different from those stored in the case library).
- CBR may introduce a reasoning bias (linked to the "best match" selected from the case library) that often leads to a non-optimal outcome.
- The quality of the solutions proposed by a case-based reasoning system is very much influenced by quantity and diversity of the set of cases available in the case library.

3.5.1. CBR versus rule-based reasoning

There are some striking similarities between rule-based and case-based systems (Barletta, 1991). Cases are indexed according to carefully chosen features that may resemble rules, and, frequently, they are generalized to eliminate irrelevancies and to limit the scope of the search.

Kolodner (1993) considers that, at some level, we can think of case-based reasoning as a type of rule-based reasoning in which the rules are very large, the antecedents need to be only partially matched, and the consequents need to be adapted before they are applied. A case's problem description might correspond to a rule's antecedent and the solution to its consequent. Rather than requiring a perfect match, we can apply the case when the antecedent partially matched. Despite those similarities, there are obvious differences between the two methods as listed in Table 3-1.

3.5.2. CBR versus pattern recognition

Both case-based reasoning and pattern recognition use a common methodology, where finding the solution is based on an initial set of exemplary cases, each case being labeled as belonging to

a particular class. As in pattern recognition, cases are identified by their "features" and classified accordingly. What pattern recognition calls "classification" or "recognition," case-based reasoning calls "matching". Beyond similar terminology, however, there are some significant differences as listed in Table 3-2.

Table 3-1. Case-based reasoning versus rule-based reasoning

	Case-Based Reasoning	Rule-Based Reasoning
1. Size	Cases are large chunks of domain knowledge that usually overlaps with other cases information.	Rules are desired to be fairly small, independent, and consistent pieces of domain knowledge.
2. Meaning	Cases in case library are constants.	Rules in a rule base are patterns.
3. Matching	A case could be retrieved if it matches the input partially.	A rule can be "fired", if and only if, its antecedent matches the input exactly.
4. Usage	Problem is solved almost at once by retrieving a case approximating the entire solution then adapting it.	A problem is solved step-by-step by applying rules in iterative cycles of micro-events controlled by an agenda.
5. Flow	For a given input set, the retrieved case should be always the same, no matter how many other cases exist in the case library, and independent of the storage order of those cases.	For a given problem, the firing order of "heated" rules (that is, rules enabled for firing) depends on the inference mechanism, their ranking and storage placement in the rule base.

Table 3-2. Case-based reasoning versus pattern recognition

	Case-Based Reasoning	Pattern Recognition
1. Matching	Usually, the "best-match" case is only an approximation of the final solution, later steps adapting and refining it.	Usually, the retrieved pattern is the final solution (with no attempt to modify a pattern, if it does not fit all the input requirements).
2. Retrieval	Cases are retrieved if they partially match the problem at hand.	Patterns are retrieved based on some probabilistic assessment of their feature sets.
3. Adaptation	A partially matching case could be adapted and repaired till it fits the input problem (if possible).	Cases that do not fit the input requirements are rejected.

3.6. CASE-BASED REASONING IN DESIGN

The range of reasoning tasks that can be supported by a case library are roughly of two types:

(a) *problem-solving tasks*, including design, planning and diagnosis, and

(b) *interpretive tasks*, including understanding, justification, and classification.

In design, problems are defined as a set of constraints, and the problem solver is required to provide a concrete solution that accommodates all those constraints. Usually, for a set of given constraints, there are many possible solutions (the constraints are said to under-specify the problem). There are situations, however, when the problem is over-constrained, that is, no solution could be found to fulfill all input requirements, therefore, some of the less important constraints should be relaxed (i.e. compromised).

Solving the problem by adapting an old solution allows the problem solver to avoid dealing with many constraints, and keeps him from having to break the problem into pieces needing recomposition. Case-based reasoning, therefore, is suitable to solve design problems that are barely decomposable (such as building-envelope design problems). Rather than solving those problems by decomposing them into parts, solving for each, and recomposing the parts, a case suggests an entire solution, and the pieces that do not fit the new situation are adapted.

Though considerable adaptation might be necessary to make an old solution fit a new situation, this methodology is preferable to generating a new solution from scratch when there are many constraints and the solutions to parts of the problem are not easily recomposed. For example, in the case of building-envelope design, given a set of building materials where each material satisfies an unique requirement of the building envelope, such as structural load, water and thermal resistance, and so forth, it is not always possible to combine those materials into a

viable building envelope (or in other words, solving the building-envelope design problem through decomposition is not always practical).

The experience a designer has in "framing" a problem could make all the difference from an ordinary to an outstanding design. This observation leads to the conclusion that case-based reasoning has a lot to offer in solving design problems, but its outcome is very much linked to the understanding of the problem, including ranking the design priorities and carefully indexing the previous cases along some significant features.

3.7. HISTORICAL PERSPECTIVE AND PREVIOUS WORK

The last fifteen years have seen a growing interest in case-based reasoning research, primarily in fields such as law, medicine, and strategic planning, where large libraries of cases have been collected.

The case-based reasoning methodology can be considered as an evolved and refined form of the nearest neighbor classification of pattern recognition. The theoretical framework of case-based reasoning is influenced by Schank's work on Memory Organization Packets (MOP), which is one of first forms of memory representation of cases (Schank, 1982). Schank's ideas were further developed by his students, most notably Janet Kolodner (1989, 1993), and Kristian Hammond (1989). Some of the most significant case-based reasoning systems developed from 1980 onwards are summarized in Tables 3-3 and 3-4.

Table 3-3. Landmark CBR systems

Year	System Name	Domain	Reference
1983	CYRUS	Information retrieval	Kolodner- 1983
1985	MEDIATOR	Dispute mediation	Simpson - 1985
1988	PERSUADER	Conflict resolution	Sycara - 1988
1989	CHEF	Cooking planner	Hammond - 1989
1989	PROTOS	Medical diagnosis (auditory diseases)	Bareiss - 1989
1989	CASEY	Medical diagnosis (heart failures)	Koton - 1989
1989	KRITIK	Mechanical design	Goel - 1989
1991	HYPO	Law interpretation	Ashley – 1991
1991	CREEK	Mud diagnosis (oil-well drilling)	Aamodt – 1991
1991	PATDEX	Technical diagnosis (CNC machines)	Richter, Weiss –1991
1992	JULIA	Meal planning	Hinrichs, 1992
1992	CABARET	Law interpretation (continues HYPO)	Skalak, Rissland –1992
1992	CLAVIER	Lockheed: Autoclave loading configurator	Hennessy, Hinkle –1992
1992	CADET	Design of electromechanical devices	Sycara et al. – 1992
1993	BOLERO	Medical diagnosis	Lopez, Plaza – 1993
1995	INRECA	Decision support for industrial systems	Althoff, Auriol, Weiss (www.inreca.org)
1997	CARICA	Managing forest fires emergencies	Terral, Avesani, Ricci (sra.itc.it/projects/carica)
1998	COMPOSER	Assembly sequence planning design	Purvis, Pu –1998

3.7.1. CYRUS

CYRUS (Kolodner, 1983) was based on Schank's dynamic memory model and MOP theory of problem solving and learning. It was basically a question-answering system with knowledge of the various travels and meetings of former US Secretary of State Cyrus Vance.

3.7.2. MEDIATOR

MEDIATOR (Simpson 1985, Kolodner and Simpson, 1989) is one of the first case-based reasoning systems, and it was developed to solve problems in the domain of dispute mediation. MEDIATOR attempts to offer solutions to disputes by assessing the new situation, and matching

it against its case library. By including a diagnosis of reasoning errors and repair cycles, MEDIATOR outlined the basic architecture of a CBR system.

3.7.3. CHEF

CHEF (Hammond, 1989) is a case-based planner in the domain of recipe creation. Given a set of planning goals, it assesses a situation by remembering cases that can help it anticipate what might go wrong. CHEF is an early example of including in the case library not only previous successful solutions but also some past failures that should be avoided in the future.

3.7.4. PROTOS

Bruce Porter and his group at the University of Texas, Austin, attempted to address the machine-learning problem of concept learning for classification tasks (Porter, Bareiss and Holte, 1990). This lead to the development of the PROTOS system (Bareiss, 1989), which emphasized on integrating general domain knowledge and specific case knowledge into a unified representation structure.

3.7.5. CASEY

CASEY (Koton, 1989) is a case-based diagnosis system for providing explanations regarding heart disease symptoms. A CASEY diagnoses patient by applying model-based matching and adaptation heuristics to the case that it has available. CASEY is a demonstration of how to use case-based reasoning for optimizing the performance of an existing knowledge-based system, where the domain (heart failure) was described by a deep, causal model. In CASEY system a

case-based and a deep model-based reasoning are combined to produce an effective expert system in significantly less time.

3.7.6. HYPO and CABARET

Another early significant contribution to CBR was the work by Edwina Rissland and her group at the University of Massachusetts, Amhearst. With several law scientists in the group, they were interested in the role of precedence reasoning in legal judgments. Cases (precedents) are here not used to produce a single answer, but to interpret a situation in court, and to produce and assess arguments for both parties. This resulted in the HYPO system (Ashley, 1991), and later the combined case-based and rule-based system CABARET (Skalak and Rissland, 1992). Phyllis Koton at MIT studied the use of case-based reasoning to optimize performance in an existing knowledge based system, where the domain (heart failure) was described by a deep, causal model. This resulted in the CASEY system [Koton-89], in which case-based and deep model-based reasoning was combined.

3.7.7. KRITIK

KRITIK (Goel, 1989, 1991) is a design problem solver for the domains of mechanical devices and electrical circuits. KRITIK creates new designs by adapting old designs while making extensive reference to a casual model describing how the old designs work. Thus, every design also holds a casual explanation of how its components work together to produce the indicated effect. While a functional representation shows how the pieces are connected to each other and shows the general effects of those connections, a casual model shows how the underlying physics allows those effects to happen.

3.7.8. JULIA

JULIA (Hinrichs, 1992) is case-based designer in the domain of meal planning. It uses cases to propose plausible solutions, decomposing the problem as necessary. Heuristic matching rules permit partial matching of types, components, and ranges while the network of previous cases is being traversed. Special-purpose structure-modifying heuristics are used for adaptation in order to meet the constraints of the current problem.

3.7.9. CADET

CADET (Sycara, Guttal, Koning, Narasimhan and Navinchandra, 1992) is a system that aids conceptual design of electromechanical devices. It consists of sub-systems called CARD (CAse-based Retrieval for Design). CADET was further refurbished and enhanced to include a device model. In its later incarnation (now called WATT), the system integrates case-based reasoning with model-based reasoning.

3.7.10. CLAVIER

CLAVIER (Hennessy and Hinkle, 1992) applied case-based reasoning to a real-world problem by trying to generate autoclave load schedules in an interactive dialogue with its user. It matches cases based on the number of parts matched and the priorities of matched parts. The user is required to manually adapt the chosen case, thus, providing an environment where the user could dynamically interact with the knowledge base.

Table 3-4. CBR systems for the construction industry

Year	System Name	Domain	Reference
1992	ARCHIE	Architectural case browser	Domeshek, Kolodner - 1992
1993	CADSYN	Analogical transformation of old cases	Maher, Zhang – 1991, 1993
1994	CASECAD	Innovative building design with AutoCAD	Maher, Balachandran - 1994
1994	SEED	Conceptual building design	Flemming - 1994
1995	BRIDGER	Bridge design	Reich, Fenves – 1993, 1995
1996	IDIOM	Spatial planning and layout design	Smith, Stalker, Lottaz - 1996
1997	FABEL	Complex architectural and building design	Gebhardt, Voss, et al - 1997

3.7.11. ARCHIE

ARCHIE (Domeshek and Kolodner, 1992) is a case-based design aid for the conceptual design of buildings developed by a research group lead by Janet Kolodner at Georgia Institute of Techology. Its main focus is to provide the user with an effective navigation tool through a maze of descriptions of previous design cases. ARCHIE emphasizes case presentation and decomposition of a large case for easy understanding by the user. The matching process uses the highest matching score as the ranking criteria, and the retrieval algorithm is based on using the nearest-neighbor approach. Archie does not generate or provide any design solutions, but serves as a reference tool for the designer. The goal was to capture and disseminate lessons learned from design experience so that future designers could avoid repeating known mistakes.

3.7.12. CADSYN and CASECAD

In CADSYN (Maher and Zhang, 1991/1993), the contribution made to the representation of cases is to have cases decomposed into subparts; by representing knowledge hierarchically in this fashion, the reuse of cases is made more flexible and efficient, since irrelevant information is

pruned from consideration for each given design problem. In CADSYN the emphasis is not on case retrieval but on case adaptation, where constraints are used as the knowledge that guides the analogical transformation of an old case to a new problem-solving context. The adaptation of a case is thus performed using a constraint-satisfaction.

In CASECAD (Maher and Balachandran, 1994), the information held in design cases is represented using multiple media: attribute-value pairs are used to indicate the specific value that the case has for different design variables; text strings are used to provide natural language explanations of certain aspects of the design; drawings are used to represent the physical appearance of the design; etc. In addition, flexible strategies for case retrieval were proposed in the CASECAD project which allow for the revision and reformulation of the design problem requirements, thus making the overall design process more innovative.

3.7.13. SEED

SEED project was developed at Carnegie Mellon (Flemming, 1994) with the goal to achieve a software environment that supports the early phases of building design and thus, in principle, to augment all aspects of the early design process that can benefit from such support. During the early phases of design of similar building types, architects tend to rely on previously established solutions and solution strategies. SEED is intended to provide a systematic computational support environment for the rapid generation of designs with respect to recurring building types. Consequently, it is necessary to enable the storage and retrieval of past solutions and their reuse as prototypical designs in similar problem situations. In this respect SEED is an application of case-based design systems. It has been envisioned that initially SEED will consist of three modules that support the following tasks: 1) architectural programming, 2) schematic layout

design, and 3) schematic configuration design. The tasks of the second and third module are, respectively, to generate two-dimensional and three-dimensional configurations of spatial elements and building components. The task of the first module, which is the primary focus of our work (SEED-Pro), is to develop an architectural program (AP) in order to initiate a design process in SEED and to support modifications and evaluations of this program.

There are many other case-based reasoning systems, covering all aspects of problem solving or problem understanding. However, their overall architecture is built on the same generic four steps: matching, adaptation, evaluation, including repair, and learning. The above selection of CBR systems included CYRUS, MEDIATOR and CHEF due to their historic value (they setup the design guidelines used by most, if not all, other case-based reasoning systems that followed), JULIA for its pioneering work to employ heuristics into a CBR system, CASEY, CADET, and CLAVIER, which demonstrate the steps of applying case-based reasoning to design, and ARCHIE which was one the first reported case-based reasoning system that addresses a building-design problem.

4. CASE REPRESENTATION FOR BUILDING ENVELOPE DESIGN

4.1. KNOWLEDGE ACQUISITION

The aim of any learning system involved in engineering design is to reach a level of expert competence. In this direction, the success of the case-based reasoning approach depends very much on the classification of cases and the quality of knowledge acquisition. The better the classification of previous experience (the indexing of cases) the more successful will be the retrieval of best match and subsequently the final outcome.

Competence of classification can be improved by combining it with explanation (Porter, Bareiss and Holte, 1990). A case is described by a collection of features but all these features must be further explained to facilitate case classification. For example, let us consider a wall (that is, a building-envelope "case") including a list of known building materials (that are "features" of the wall case). To increase the knowledge encapsulated into the wall case, the presence of each building material should be accompanied by an explanation with respect to its intended function (i.e. "polyethylene" is a vapor barrier to prevent vapor diffusion; "expanded polystyrene" is an insulation material to prevent heat losses, and so forth). Explanation adds importance to features making them mandatory, desirable, optional, or unwanted thereby facilitating the matching process.

The quality of knowledge acquisition is characterized by:

- the amount of training samples required for reaching reasonable levels of competence in classification and explanation, and
- the degree of autonomy gained by the system through knowledge acquisition. Autonomy is defined by the program-issued requests for knowledge.

The desirable behavior of a learning system is to pass from individual instances to general concepts, that is, to aid generalization by linking cases within a broader context. In building-envelope design domain, there are many possibilities for generalizations by extracting the most important features of any item. For example, a "built-up roof" is an instance of the more general concept of "roof", which may be seen as a mandatory feature of the "building" concept. That hierarchy of concepts and features, including their abstraction and interdependence, is an important part of the domain knowledge known by the system (Sycara and Navinchandra, 1989). There are two basic ways to feed such information into a case-based reasoning system:

(a) the first alternative supposes that the human expert, acting as "teacher" for the system, pre-processes the training examples so that all features of a case are clearly differentiated and their relationships explained,

(b) the second possibility is to eventually include in the system some domain theory in the form of rules or constraints.

If the first method accounts more for experience acquired in the field by the human expert, the second method relies on the general theories applicable to the field (i.e. proven physics laws), mandatory constraints imposed by legislation, generally accepted knowledge (common-sense engineering theories), handbooks, codes, and so on. Hence, the knowledge base of the system should be a combination of pre-processed cases and domain rules.

4.2. CASES AND FEATURES IN BUILDING-ENVELOPE DESIGN

4.2.1. Indexing building-envelope cases

As a prerequisite to implementing a CBR system for the building-envelope design, a reasonable number of known building-envelope cases should be available for inclusion in the system's

database. These cases are to be indexed into the case library according to their features. The indexing vocabulary used will have a major impact on the retrieval of the appropriate case for a given design context.

Research on indexing over the past decade has led to the conclusion that good indexes are features that distinguish cases based on some important attributes in the case description. A good indexing mechanism has to be driven by the concepts that are normally used to describe the items being indexed, whether they are surface features or something more abstract. For example, a given wall will be better indexed by its construction type (cavity wall, masonry wall, and so forth), which may be used as a retrieval key, than by its number of layers, which has less retrieval significance.

The following guidelines are used in choosing indexes for particular cases:

- *Index vocabulary should be a subset of the vocabulary used for the fully symbolic representation of cases* (that is, the indexes that are used to map a case in the case library are extracted from the set of features that describe the case within the case library).
- *Indexes should be predictive* by addressing the purposes for which the case will be used.
- *Indexes should be abstract* enough to make a case useful in a variety of future situations (features of a case that are useful as indexes often go beyond the surface representation of the case).
- *Indexes should be concrete* enough to be easily recognizable in future situations.

- *Indexes should provide a unique access key to a case* (that is, a given set of indexes should always lead the retrieval process to the same case, or group of cases, no matter how many cases are stored within the library.
- *Indexes should be linked to features encoding essential functional characteristics* of the case so that a mapping mechanism to transfer information between cases can be implemented.

This research anchored the choice of indexes for building envelope cases on the building features identified by the ASHRAE Standard 90.1/1989 "Energy Efficient Design of New Buildings Except New Low-Rise Residential Buildings" (ASHRAE, 1989). The Standard is based on extensive research in energy conservation in buildings, and includes new or refined principles for building design, with the capacity to recognize advances in the performance of various components and equipment. The Standard fosters innovative energy conserving designs by encouraging the designers to take into consideration the dynamics that exist between the many components of a building through the use of the System Performance Criteria or the Building Energy Cost Budget Method compliance path. Using as guidelines the recommendations made by the ASHRAE Standard 90.1/1989, the following parameters are identified as *salient features* for preliminary building-envelope design:

- *building type* (intended usage of the new building),
- *building height* (total number of floors),
- *building size* (the total floor area),
- *building aspect ratio* (the ratio of building width over the building depth),
- *window-to-wall ratio* (the percentage of glazing area of the total wall area), and

- *building location* (the average climatic data for building location).

Additional parameters (also known as *secondary features*) have an impact on the design outcome. These include:

- *building occupancy* (that is, the average number of persons per unit of floor area),
- *internal loads* (that is, the general electrical service loads including lights and equipment),
- *shading coefficient*, and
- *projection factor* (that is, the external horizontal shading projection depth).

To simplify the design methodology, but without loosing the generality of the solution, the following three assumptions are made:

- The secondary features could always be set to the values recommended by ASHRAE.
- The energy performance of each building envelope varies linearly within the scale of the climatic data. Therefore, any two building-envelope designs may be compared, with the System Performance Criteria, by simulating their relative behavior for a given location. This assumption is made to avoid storing climatic information into the case library and it can be accepted only in the context of a limited geographical area (such as that covered by the SPC equations).
- Building features with large input ranges, such as the building height and the building size, could be clustered to fit into a smaller number of discreet categories.

4.2.2. Mapping the salient features of the building envelope

As mentioned above, aiming to achieve the energy efficiency of the new building leads to selecting the salient features based on ASHRAE 90.1/1989 recommendations. In ASHRAE Standard 90.1/1989, the building type is classified according to the intended usage of the building (that is, its occupancy type) based on one of the following categories:

1) *Assembly,*
2) *Office,*
3) *Retail,*
4) *Warehouse,*
5) *School,*
6) *Hotel,*
7) *Restaurant,*
8) *Health Institution,*
9) *Residential.*

In addition to the building occupancy type, the geometry and dimensions of the new building are important parameters for classifying the cases. For instance, the height of the building determines the suitable structural types for that building, while the size of the building is a primary criterion in selecting the appropriate HVAC system. Recording the building dimension characteristics as continuous values would increase the difficulty of matching and transforming an envelope case. A digitized feature mapping is therefore recommended. Rather than recording the actual number of floors as the building-height feature, the following four building-height categories are defined:

1) *low rise* : buildings with 3 or less stories

2) *medium-low rise* : buildings with 4 to 10 stories

3) *medium-high rise* : buildings with 11 to 20 stories

4) *high rise* : buildings with more than 20 stories

Similarly, for the building's size, and guided by the ASHRAE suggested ranges for the selection of the HVAC systems, the following size categories can be established:

1) *size type 1* : 0 ft^2 ≤ total floor area < 5,000 ft^2 (< 464.52 m^2)

2) *size type 2* : 5,000 ft^2 ≤ total floor area < 10,000 ft^2 (< 929.03 m^2)

3) *size type 3* : 10,000 ft^2 ≤ total floor area < 15,000 ft^2 (< 1,393.55 m^2)

4) *size type 4* : 15,000 ft^2 ≤ total floor area < 20,000 ft^2 (< 1,858.06 m^2)

5) *size type 5* : 20,000 ft^2 ≤ total floor area < 50,000 ft^2 (< 4,645.15 m^2)

6) *size type 6* : 50,000 ft^2 ≤ total floor area < 75,000 ft^2 (< 6,967.73 m^2)

7) *size type 7* : 75,000 ft^2 ≤ total floor area (≥ 6,967.73 m^2)

4.2.3. Building-envelope cases

In building-envelope design, a case may be seen as the instance of a conceptual object called *Envelope*. The *Envelope* object might be decomposed into three smaller component objects: *Wall, Glazing,* and *Roof*. In addition to its components, an *Envelope* object has a feature vector labeling its functionality. That feature vector should include the main characteristics of the envelope such as the structural requirements, the size and the geometry of the building (total wall and roof area, building height, shape, orientation, window-to-wall ratio, and so forth), and the building occupancy type (office, residential, school, warehouse, hospital, to name a few). These features are not necessarily linked and their relative importance is quite difficult to quantify. In

general, the problem to solve is the following: given a collection of n known *Envelope* objects, such as:

$$Envelope_i = \{Wall_i , Glazing_i , Roof_i , vector_i\}$$

where $i = 1,2,...,n$, find the suitable indexing strategy to organize these objects into a case repository.

A previous knowledge-based approach to building-envelope design (Fazio et al, 1989, Gowri, 1990) has demonstrated the need for problem decomposition. In that approach, the complex *Envelope* object is decomposed into component objects; therefore, instead of storing the *Envelope* cases as complex entities described by large feature vectors, three different libraries of components are assembled: a first one for *Wall* objects, a second one for *Glazing* objects, and a third one for *Roof* objects. Case decomposition allows enforcing the economy of representation since the same component may be linked to several *Envelope* instances.

4.2.4. Building-envelope case representation

In Dynamic Memory (Schank, 1982), Schank proposed organizing structures, called *Thematic Organizational Packets (Units)*, or TOP, as organizers of cases that are thematically similar to each other, that is, those that share common goals, plans, and goal/plan interactions.

The building-envelope cases can be represented according to the TOP concept by defining a set of frames (or classes), each frame being the class abstraction for groups of envelope components. Cases are then represented along two types of specifications:

i. the **Content Grid** is the representation mechanism that captures the content of that case (such as the layers included in an envelope element), and

ii. the **Context Grid** is that part of the case representation keeping track of relationships between cases (such as their suitability for a given building type, structure, height, size, and so forth).

While the content grid is linked to the composition of envelope elements, the context grid is linked to the indexing mechanism that designates under which circumstances the case should be retrieved.

Library traversal is possible through links established between frames, and, at a given frame level, through links between instances.

4.3. LIBRARY OF BUILDING-ENVELOPE CASES

There are two approaches to establish the set of indexes suitable to encode library cases:

- *Functional Approach*:

  The corpus of available cases is examined (often through computerized means) and some similarity patterns are established.

- *Expert Approach*:

  A human expert selects a set of features most likely to be considered as relevant for a case.

4.3.1. Building vector

Following the recommendations put forward by the ASHRAE Standard 90.1, the building attributes identified as primary features for a building-envelope case (such as building type, height, size, aspect ratio, and window-to-wall ratio) are assembled into a *building vector* that will be further used to bundle together building-envelope cases:

Building Vector = {Type, Height, Size, Aspect Ratio, Window-to-Wall Ratio}

The same vector could be matched by a large number of buildings, therefore, to implement an effective reasoning methodology, it is necessary to further differentiate the cases within the building class represented by a given building vector. Moreover, it is often needed to bypass the building categories when looking for specific features. One approach to the case classification problem is to adopt an additional clustering mechanism based on the *physical signature* of the building envelope that is described later.

4.3.2. Encoding a building-envelope case

The encapsulation model used for encoding a building-envelope case is related to the amount of relevant data required to fully describe the case while preserving a certain level of abstraction. A compromise must be made between the wealth of information required to make the case unique, and the need to keep the retrieval and matching process within reasonable limits of computation time.

In the proposed model, a building-envelope case consists of three major parts (or sub-cases): *wall*, *roof*, and *glazing*. These basic parts are described along the constituent layers of building materials. Each layer is labeled according to its specific functionality as belonging to one of the following nine functional categories:

1) *cladding*: cladding for walls and exterior face for roofs
2) *cavity*: air cavity for walls and glazing
3) *coating*: coating and sheathing
4) *membrane*: vapor barriers, air barriers, roof water proofing
5) *panel*: exterior panels for walls

6) *insulation*: all type of wall and roof insulation (rigid or non-rigid)

7) *structure*: wall structures (including studs), roof framing, and decks

8) *finish*: interior finish, wallboards

9) *other*: glazing, glass blocks, gravel, mortars, and so on

Moreover, for each layer of building material, the case records a relevant set of physical parameters such as thickness, density, thermal resistance, vapor resistance, specific heat, etc. To keep the case description at a minimum, details such as joints, appearance, durability, maintenance costs, and so on, are not included. The building vector that will be used as the primary classification criteria augments the case information. As an example, Table 4-1 shows the simplified description of a building-envelope case sample (without including the building vector and the information regarding the physical properties). The roof and wall subsystems for the building envelope case of Table 4-1 are graphically shown in Figures 4-1 and 4-2 respectively.

Table 4-1. Example of a building envelope case

Masonry Cavity Wall		**Double Glazing**		**Hollow Core Roof**	
Gypsum board	*Panel*	Glass	*Other*	Gypsum board	*Panel*
Furring	*Other*	Air space	*Cavity*	Furring	*Other*
Concrete block	*Structure*	Glass	*Other*	Hollow core slab	*Structure*
Polyethylene sheet	*Membrane*			Polyethylene sheet	*Membrane*
Glass fiber R-16	*Insulation*			Polystyrene	*Insulation*
Air space	*Cavity*			Roofing	*Membrane*
Brick	*Cladding*			Stone ballast	*Other*

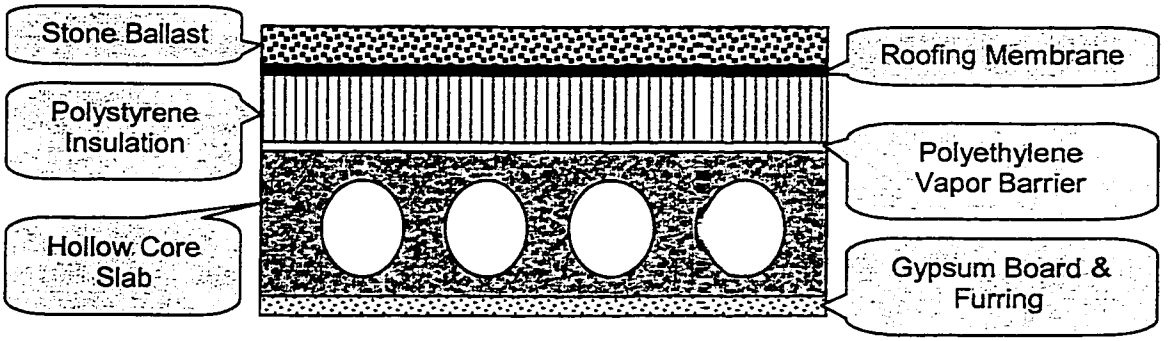


Figure 4-1. Hollow Core Roof

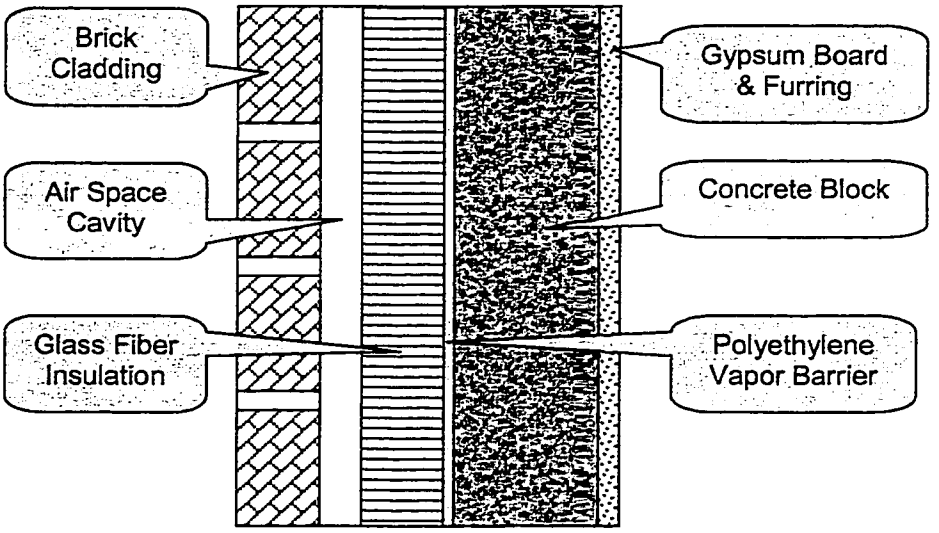


Figure 4-2. Masonry Cavity Wall

4.3.3. Classes of envelope components

An extensive analysis (including computerized pattern-recognition methods) of building-envelope cases lead to the conclusion that the *Wall* and *Roof* components can be classified further into the following generic categories:

- Five *Wall* classes:
    1) *Non-Cavity Wall*
    2) *Masonry Cavity Wall*
    3) *Curtain Wall*
    4) *Framed Wall* (a.k.a. stud walls). The stud walls can be further classified according to the level of protection provided to the air cavity into the following two types:
        - *Type 1: Exposed Air Cavity* (the air cavity the is facing directly the cladding layer)
        - *Type 2: Protected Air Cavity* (the air cavity layer is protected from the outdoor condition by at least one additional layer such as an air barrier or a sheathing layer).
- Four *Roof* classes:
    5) *Low-Pitch Wooden Roof*
    6) *High-Pitch Wooden Roof*
    7) *Metallic Roof*
    8) *Concrete Roof*

Figure 4-3. Wall Classes

Figure 4-4. Roof Classes

The class identifiers used for each *Wall* or *Roof* type are mostly symbolic and conform to the current terminology that can be easily understood by the designers. As examples, Figure 4-2 shows an instance of a *Masonry Cavity Wall* class, while Figure 4-1 displays an instance of a *Concrete Roof* class. Figures 4-3 and 4-4 detail the description of each envelope component type. It should be noted that layers marked with (*) are optional. Attaching a class label to an envelope component will facilitate envelope case matching and enable a better case-transformation mechanism.

4.3.4. Embedded knowledge-base hierarchy

A building-envelope case has an embedded object hierarchy structured from top to bottom as follows:

- *Building envelope*
- *Envelope components*
- *Building material layers*
- *Physical properties of building materials*

Figure 4-5 details this knowledge-base hierarchy and outlines the building-envelope features at each decomposition level.

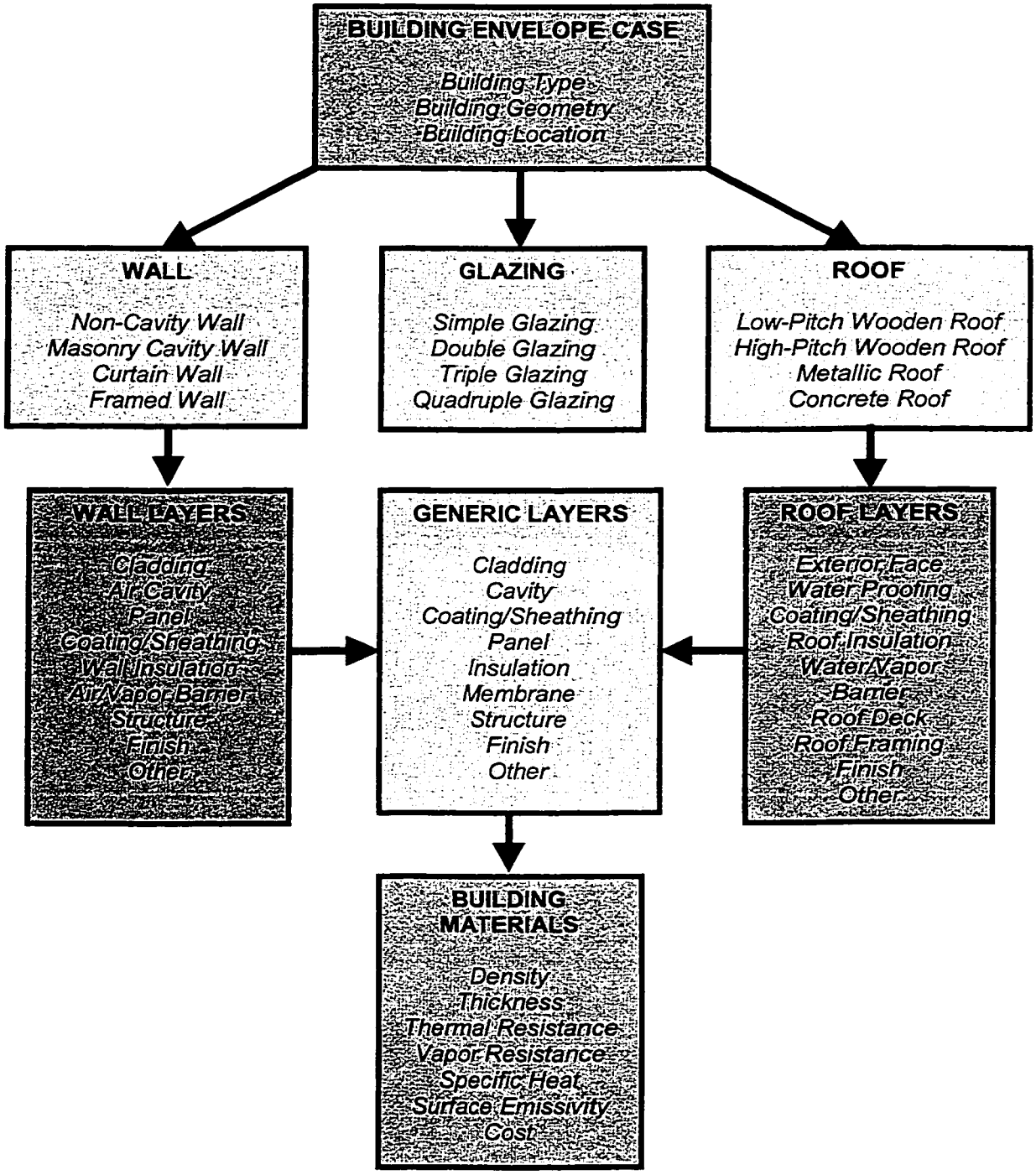


Figure 4-5. Hierarchy of Building Envelope Objects

4.4. CLUSTERING BUILDING ENVELOPE CASES

While the selected salient features could be useful in classifying groups of building cases, they are not adequate for case selection. To overcome this limitation, a secondary selection criterion is employed for matching and transforming library cases based on similarity metrics. Along that direction, the concept of the *physical signature* was developed for a building-envelope case. Each building envelope case is associated with a numerical pattern that describes the envelope cross-section along a set of predefined physical parameters. The idea behind the physical-signature concept is to define a mapping mechanism between the building envelope case and its cross-section pattern, so that the building-envelope case classification and retrieval problems can be further addressed through pattern recognition techniques.

4.4.1. "Physical signature" of the building envelope

The parameters selected to define the physical signature of the building envelope were among those with a significant impact on the compliance checking process. Therefore, without limiting the generality of the solution, the selection was confined to four parameters as follows:

1) *structural characteristics*,
2) *heat capacity*,
3) *thermal resistance*, and
4) *vapor resistance.*

The heat capacity and the thermal resistance are critical for checking the envelope compliance with the energy efficiency requirements, while the vapor resistance plays a major role to avoid moisture condensation within the envelope.

The physical signature of the entire building envelope is assembled out of the physical signatures of its main components: the wall component and the roof component. To avoid using the absolute numerical values for properties that might vary within a large range, a normalization technique was adopted to limit a signature value between 0 and 1. Each physical signature is a two-dimensional vector, one dimension associated with the normalized value, and the other one with the normalized relative position of the signature. For an envelope component (that is, a wall or a roof) having n layers, the *signature vector* associated with the physical parameter k (where k is 1, 2, 3 or 4) is defined as follows:

$$SignatureVector^k = \{SignatureValue^k, SignaturePosition^k\}$$

$$SignatureValue^k = \frac{\max_{i=1}^{n}(ParamValue_i^k)}{OverallParamValue^k}$$

$$SignaturePosition^k = \frac{(x_{ext}^k + x_{int}^k)/2}{OverallThickness}$$

The *signature value* associated with parameter k is defined as the number representing the maximum value encountered for that parameter in any of the n layers of the envelope subsystem divided by the overall value of parameter k for the whole envelope subsystem. The *signature position* is the relative depth within the cross-section of the envelope component where the signature value reaches its maximum level (x_{ext} and x_{int} are the cross-section coordinates of respectively the outer and the inner borders of the maximum level area). Usually, a physical signature points toward the layer (called *anchor layer*) having the most impact in the overall

performance of the envelope with respect to a given physical characteristic. As an example, Table 4-3 shows the physical signature values for the sample wall of Table 4-2.

4.4.2. Case filtering

The physical-signature concept was defined to enable a technique for mapping the envelope cross-section to a numerical vector. This vector can be considered as the digital footprint of the envelope, and it can be further used to assimilate the case ranking as a pattern-recognition problem.

Table 4-2. Wall sample

Concrete Wall		**Material Properties**			
Layer (Inside -> Outside)	Type	Thickness (mm)	Heat Capacity (kJ/m^2C)	R-value (m^2C/W)	Vap. Res. (m^2secPa/ng)
1. Gypsum board	*Panel*	16.00	*11.52*	*0.10*	*0.00060*
2. Concrete block	*Structure*	190.00	*0.00*	*0.20*	*0.00730*
3. Polyethylene sheet	*Membrane*	0.15	0.39	0.00	0.29000
4. Glass fibre	*Insulation*	51.00	5.88	1.54	0.00029
5. Air space	*Cavity*	25.00	35.21	0.18	0.00000
6. Asbestos Cement	*Panel*	6.00	9.12	0.01	0.05200
7. Parging Cement	*Cladding*	13.00	8.74	0.02	0.00400
Overall Wall Properties		**301.15**	**70.80**	**2.21**	**0.35000**

Table 4-3. Physical signatures for a wall sample

Signature Type	**Anchor Layer**	**Signature Value**	**Signature Position**
Structure	2 (Concrete block)	**1.00** (1.0/1.0)	**0.37** (016 + 206)/(2*301)
Heat Capacity	5 (Air Space)	**0.50** (35.21/70.8)	**0.90** (257 + 282)/(2*301)
Thermal Resistance	4 (Glass fiber)	**0.70** (1.54/2.21)	**0.77** (206 + 257)/(2*301)
Vapor Resistance	3 (Polyethylene)	**0.83** (0.29/0.35)	**0.68** (206 + 206)/(2*301)

One classification method employed by pattern recognition is the nearest neighbor approach, where the similarity of two patterns is assessed based on the Euclidean distance between the feature vectors of the patterns. For instance, the Euclidean distance between the physical-signature vectors of two envelope cases, A and B, is defined as follows:

$$DIST_{A \rightarrow B} = \sqrt{\sum_{i=1}^{n} \left(A_i - B_i\right)^2}$$

The maximum size "n" of the physical signature vector for the entire envelope is 16 (4*2 for both wall and roof). If the building-envelope case is considered to be described by the building vector (whose size is 5), and the physical signature is the set of features extracted from the case, then the matching problem is reduced to the mapping of two vectors: the building vector as input, and the physical signature as output. This type of problem can be solved by an *artificial neural network* (ANN) implementation. An ANN can be trained to accept as input the building vector and to deliver as output the physical signature vector of the envelope (Table 4-4). Given a comprehensive set of building-envelope cases, following a proper selection of the network topology, and after the appropriate training procedure, the ANN is supposed to associate with each building class a generic physical signature (also known as *class signature*).

Table 4-4. In/Out vectors of the ANN-based case filter

Input: Building Vector	**Output: Signature Vector**	
	Wall	**Roof**
Building Type		
Building Height Category	*Structural Type*	*Structural Type*
Building Size Category	*Heat Capacity*	*Heat Capacity*
Aspect Ratio	*Thermal Resistance*	*Thermal Resistance*
Window-to-Wall Ratio	*Vapor Resistance*	*Vapor Resistance*

To outline how the ANN filter is intended to work, let us suppose that the case library has, under a given building class, only three building-envelope cases: A, B, and C. In other words, all three cases fit the same general description of the building: same type, same height, same size, and so forth. If a simple matching function is employed to further classify the three cases, then all of them will yield the same score. Now, if the above ANN is trained to match the three envelope cases with their physical signatures, then further ranking can be achieved by comparing the case signatures against the class signature. The advantages of using that clustering mechanism are as follows:

- Buildings within the same class (i.e. same building description) can be differentiated.
- A preliminary description of the building envelope is already provided (the physical signature is to a certain extent an image of the building-envelope cross-section).
- When a suitable match is not found within a given class, the physical signature is a good starting point to initiate the searching procedure that will go beyond the class boundaries.

It should be noted that the physical signature method is based entirely on heuristic knowledge.

5. CBR FRAMEWORK FOR BUILDING ENVELOPE DESIGN

5.1. RETRIEVING A BEST-MATCH CASE FROM CASE LIBRARY

The process of retrieving the most suitable design alternative for a given design context is called *case matching*.

The retrieval of cases from a case library can be seen as a massive search problem, but the search process in case-based reasoning could result in a close partial match rather than a full match. No case in the library can ever be expected to match a new situation exactly, so a search will always result in the retrieval of the closest possible match. Partial match algorithms are quite expensive (that is, complex and time consuming), and, therefore, the retrieval must be directed in some way so that matching is only attempted on those cases with potential for adaptation to a new situation.

Algorithms for searching a case library are associated with the organizational structure of the library. When a case library is large, there is a need to organize cases hierarchically so that only some small subset needs to be considered during retrieval. This subset, however, must be likely to have the best matching or most useful cases in it.

The two basic types of case library organization are (i) the *flat model*, when cases are placed in a linearly linked list, and (ii) the *network model*, when cases are hierarchically organized. A recommended approach is to have a mixed organization, combining both the flat and the network models. Then, a breadth-first initial search, to delineate all potential cases, is followed by a case filtering procedure to narrow down the matching scope to a smaller subset of cases that are the most likely to fit the target requirements.

5.1.1. Assessing similarity between building envelope cases

Assessing whether a building envelope case is similar to a new design context requires determining whether other envelope cases are more similar. Comparing two envelope cases, C_i and C_j, with respect to a given set of design requirements can be graphically represented in terms of Venn diagrams as shown in Figure 5-1. The comparison should be performed along the case attributes relevant to the design outcome.

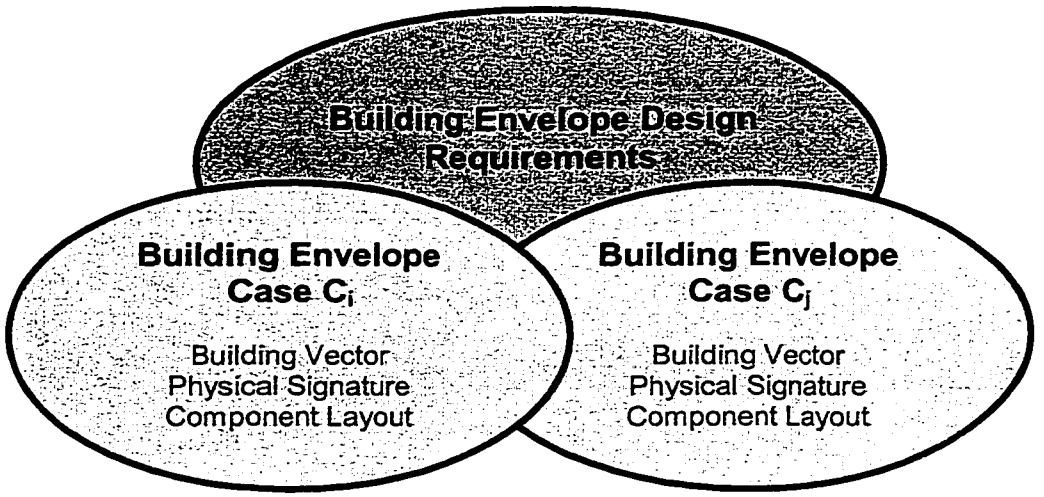


Figure 5-1. Comparing Building Envelope Cases

Three basic methods have been identified (Ashley, 1989) for ranking the relative importance of individual features in the analysis of cases: 1) *analytical*, 2) *statistical* and 3) *precedent-based.* The analytical model defined for the building envelope (and shown in Figure 4-5) can be used to infer the relative importance of some features. For instance, the structural characteristics of an envelope case should be considered more important than its appearance-related attributes. The statistical analysis of cases can lead to assigning numerical weights to features and thus to

discriminate among them. For example, the physical signature ranking can be considered a statistical method. The precedent-based methods address the need of ranking competing and/or conflicting attributes. The ASHRAE-based building vector is an example of precedent-based approach.

5.1.2. Matching strategy

Selecting the appropriate matching strategy has a significant impact on the design outcome. The strategy adopted here has three steps as follows:

- Select a *list of suitable design alternative cases* by using the building vector as the primary matching criteria.
- Cases sharing the same building vector are ranked according to the Euclidean distances between their physical signatures and the class signature.
- The matching metrics is augmented with a cross-section analysis component.

5.1.3. Defining a subset of suitable design alternatives

To define the subset of suitable design alternatives, all cases in the library are ranked according to an *evaluation function* having the following form:

$$F_{T,C} = \frac{\sum_{i=1}^{n} [w_i \times compare(f_i^T, f_i^C)]}{\sum_{i=1}^{n} w_i} \Rightarrow \sum_{i=1}^{n} [w_i \times compare(f_i^T, f_i^C)]$$

where w_i is the weight associated with feature i, while f_i^T accounts for the value of feature i in the target case T (our design goal), and f_i^C is the value of feature i in the library case C. If the

evaluation function yields a value above a predefined threshold, then the case at hand can be considered to be part of the list of suitable design alternatives.

The list of features to be included in the evaluation function has to be carefully determined to allow a comprehensive and meaningful selection of cases. While the evaluation function should filter out all non-practical solutions, no suitable alternative is to be discarded. The list of features recommended for the evaluation function should include the salient features of the case such as:

- Building occupancy type
- Building geometry (height, size, and aspect ratio)
- Window-to-wall ratio
- Generic class type for the wall, glazing, and/or roof components

5.1.4. Filtering cases

The System Performance Criteria of ASHRAE Standard 90.1/1989 allows building designers to assemble a theoretical model of the building, called the *prototype building,* as the target to meet in terms of energy efficiency for a given building type at a given geographical location (see Appendix A for additional details).

The *prototype building* defines the heating and cooling criteria to be met by the new building design. Among the parameters with an important impact on the calculation of the heating and cooling loads are some of the overall physical parameters such as the total thermal resistance and the total thermal mass of the building envelope; consequently, those physical parameters are prime candidates for defining the physical signature of the envelope.

The physical signature of the new envelope design can be used to rank similar cases in the list of suitable design alternatives, cases that otherwise would have been equally recommended. Filtering cases based on the physical signature is possible only if all library cases record a physical signature for the target geographical location. Figures 5-2 and 5-3 present how case filtering based on the physical signature is made possible.

First, as shown in Figure 5-2, a back-propagation neural network is trained to match the building vectors of library cases with their associated physical signature vectors (computed for a target location).

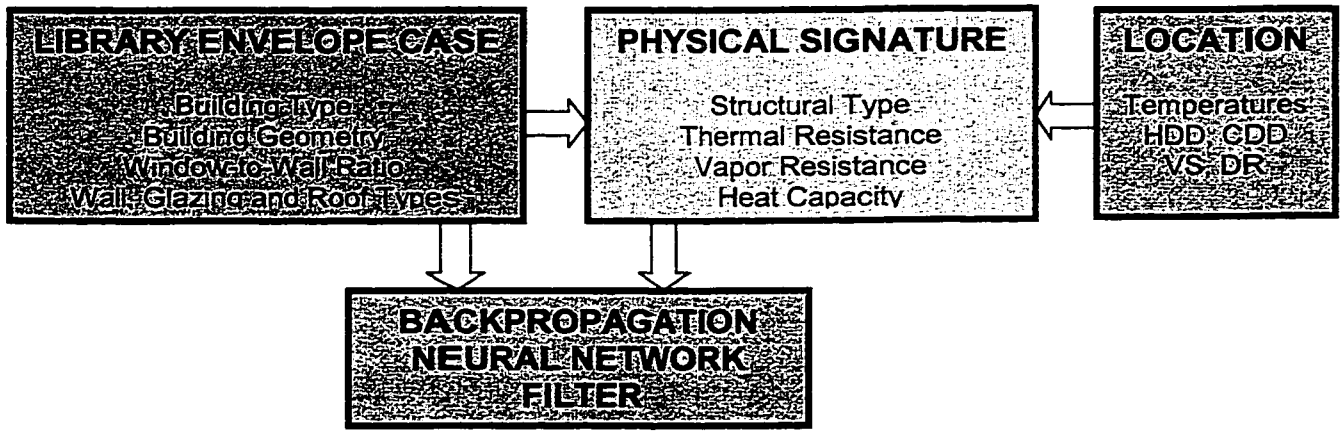


Figure 5-2. Training ANN Filter

The normal outcome of the training procedure should be the clustering of envelope cases into classes. An average physical signature called *Class Signature* characterizes each class.

Next, as shown in Figure 5-3, the new envelope case is processed by the ANN to determine its matching class signature.

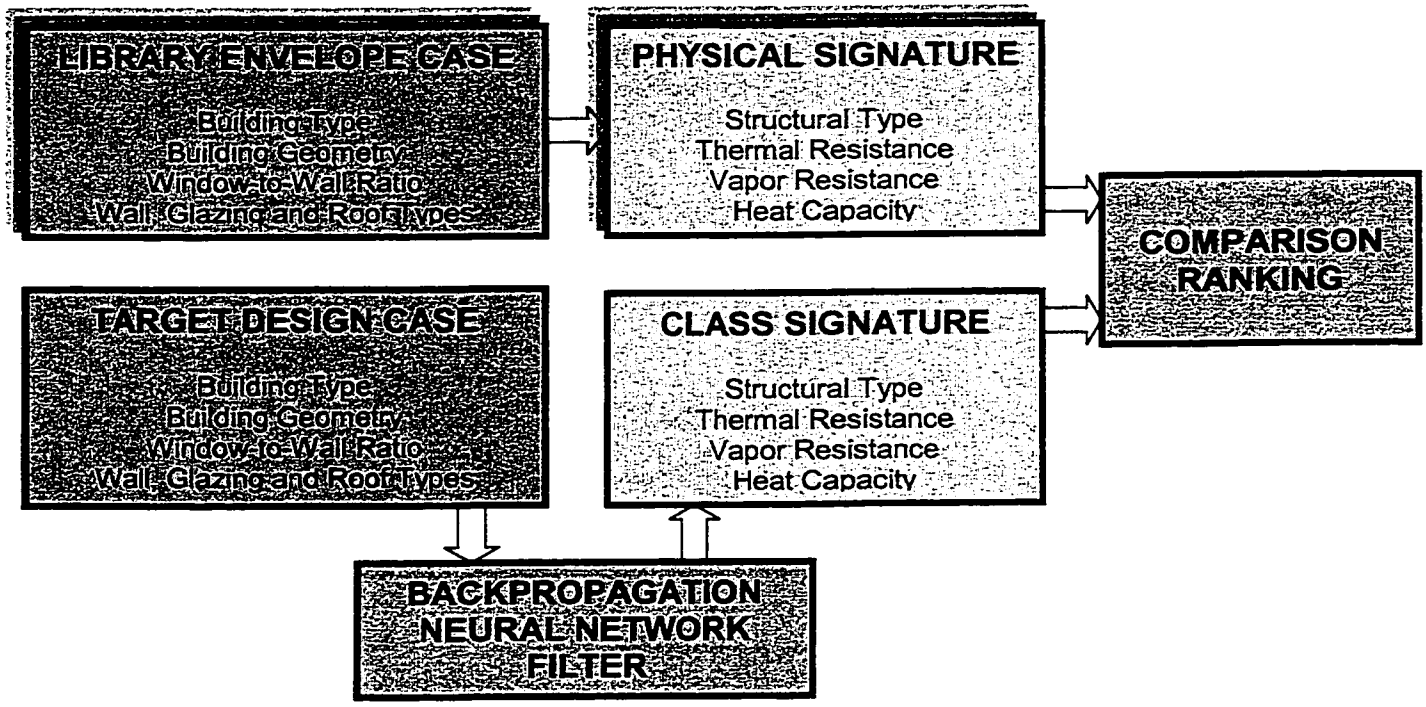


Figure 5-3. Case Filtering

Finally, the class signature of the new case is compared against the physical signatures of cases in the list of suitable design alternatives, and those cases are further ranked accordingly.

5.1.5. Matching algorithm

The case matching procedure includes the following steps:

I. Define two matching thresholds *n1* and *n2* (*n2* < *n1*), where *n1* is the maximum number of cases to be retained for matching analysis, and *n2* is the maximum number of cases to be considered as suitable design alternatives for the current requirements.

II. Browse through the entire case library to identify building cases suitable for the current design context.

III. Rank the list of suitable cases according to their building vectors. Rank cases sharing the same building vector according to their physical signatures.

IV. Apply the matching criteria to the *n1* building cases having the highest-ranking score.

V. Retain the first *n2* cases with the highest matching score as *suitable design alternatives*.

VI. Label the top case in the list of suitable design alternatives as *base building case*. It should be noted that, initially, the base case is actually the *best match case*.

5.1.6. Cross-section analysis

The case-matching outcome can be improved without adding more complexity to the building-envelope case description by including a cross-section analysis anchored on a lexical content-parsing mechanism.

First, a list of meaningful keywords is identified. The keywords in the list should have either a functional meaning (such as "*DECK*", "*CURTAIN*", "*PANEL*", "*MASONRY*") or are marked as structural qualifiers (such as "*WOOD*", "*STEEL*", "*CONCRETE*", "*BRICK*", and so forth).

Second, the keywords identified in the design specs are compared with the keywords found in description of the library case currently under scrutiny, and the degree of similarity is assessed based on lexical content matching.

This lexical analysis strategy will limit (if not eliminate altogether) the need to expand the case description with more information such as the structural and/or the construction model. The cross-section analysis is completed by a layer-type checking where the matching score is increased for the library cases that fit the cross-section description of the design case along certain layer types. Moreover, additional matching weight is associated with layers difficult to replace such as the structural and/or cladding layers.

5.2. CASE ADAPTATION

The retrieval process yields a set of potentially useful cases, all of which partially match the new situation. Through matching and ranking, we achieve the best case to start to reason with. Next, we need to replace those unmatched parts to meet our design requirements. That step is called *case adaptation*, or transformation. Features in the target case are mapped into those of the retrieved case by a transformation technique that works along similar features (i.e. features that could be swapped without a change in functionality or general characteristics).

The case transformation is required to adapt the base case to fit as close as possible the design requirements. The outcome of that process is the *modified building case*, a case preserving the usefulness of the base case while exhibiting a different set of features. The transformation process might involve major steps such as the replacement of an entire envelope component (that is, either the wall or the roof), or minor modifications such as changing a given

layer. It might even include the re-instantiation of the retrieved case by looking into the case hierarchy at its class abstraction.

The main challenge of implementing a set of reliable adaptation procedures lies in the lack of well-defined theoretical rules. Therefore, one approach is to adopt as transformation strategy a technique that relies on the information encapsulated in the case library. The envelope elements are to be replaced through an iterative process during which the adjacent elements are analyzed as well. For instance, before replacing a given layer "*x*" with another layer "*y*," the following analysis steps are to be performed:

- Both layers "*x*" and "*y*" should have the same functional role within the envelope component.
- The layers currently adjacent to "*x*" (i.e. "*x+1*" and "*x-1*") should be acceptable neighbors for layer "*y*". A case library-wide search can identify those instances where "*y*" is paired with "*x+1*" and/or "*x-1*".
- By swapping layer "*x*" with layer "*y*," the current envelope case should better fit the design requirements. In other words, a layer is not replaced unless there is a benefit in doing so.

The same approach is employed for replacing entire envelope components. For instance, before replacing either the roof or the wall, the CBR system has to check that the two elements can be paired. Transforming the base case into the modified case follows several steps:

I. Define the similarity threshold to be used as the criterion for ending the adaptation process.

II. Select as source (or base) case the case currently placed at the top of the suitable design-alternatives list.

III. Establish the design requirements not currently matched by the base case.

IV. Assign the transformation priorities. Since the System Performance Criteria checks more thoroughly the wall performance than the roof compliance, a lower priority is assigned to roof transformation than to wall transformation. Moreover, the layers more difficult to match (such as the structural layer), are to be replaced first, and are assigned a higher priority.

V. Modify the base case towards conforming to current design context.

VI. Assemble the modified case as a version of the base case that conforms to the design requirements within the selected similarity threshold.

VII. Verify, through a case library-wide search, that the modified case is a valid design alternative: check the current roof-wall pairing, and check the coexistence of adjacent layers.

If the case transformation failed, then the current base case is removed from the list of suitable design alternatives, and the transformation procedure is restarted.

5.3. CASE TESTING

The modified cased should comply with the local construction practices, and it should meet the predefined performance level. Envelope performance usually relates to the thermal performance of the envelope and the control of the moisture migration through the envelope. The thermal performance of the envelope can be assessed through the heating and cooling equations of the System Performance Criteria from ASHRAE 90.1/1989 (see Appendix A), while the potential for moisture induced problems can be evaluated through the well-known building-science equations (see Appendix E).

The well-known construction practices and the building standards can be encoded in a set of knowledge-based rules such as:

- The vapor barrier should be placed on the warmer side of the insulation.
- If there is more than one layer of insulation, then the vapor barrier could be sandwiched between insulation layers, on the warmer side of the rigid insulation (as recommended in Section 2.2).

It is obvious that, the above rule should be applied only if, and only if, the envelope component has two different layers of insulation (one rigid and the other one non-rigid).

The envelope performance is checked through numerical simulation, and all the following three conditions should hold:

- No concealed and/or surface moisture condensation occurs.
- The thermal resistance of the roof should be within the limits recommended by ASHRAE (see Appendix B).
- The wall should pass the System Performance Criteria for energy efficiency.

If the testing is successful, then the modified case is labeled as the *verified building case*; otherwise, on failure, the case is marked as needing further repair.

5.4. CASE REPAIR

Whenever a modified case does not meet the design requirement and/or constraints, it has to be repaired. A design failure is usually linked to either a thermal performance below the acceptable level or to the potential for moisture condensation within the envelope.

The same procedures and rules employed to check an envelope case can be expanded to be used for repairing it. It is, however, possible that the repair procedure fails as well. For

example, by trying to increase the thermal insulation of a wall, we might exceed the upper limit established for the wall thickness. The information sources available for defining a set of repair rules are as follows:

- building-science equations,
- ASHRAE recommendations,
- local construction requirements, and
- well-established construction guidelines.

Based on that information, a knowledge-based engine for case repair could be assembled. The engine would rely on a set of rules to repair a case that fails the validation tests. For example, some of the rules used in the attempt to repair an envelope case that exhibits moisture condensation problems are as follows:

- Increase the vapor resistance.
- Modify the position of the vapor barrier within the wall and/or roof.
- Modify the thermal resistance.

Similarly, an envelope case failing the energy efficiency tests is to be repaired by using rules such as:

- Increase the thickness of the thermal insulation layer.
- Modify the type of the thermal insulation for increased protection.
- Reduce the window-to-wall ratio below the design requirements.

The repair could fail if the simulation tests are not passed, while the case is modified beyond a given threshold. For instance, the attempt to repair an envelope case is stopped if any one of the following conditions occurs:

- the overall wall or roof thickness exceeds the design limits,
- the window-to-wall ratio is reduced below a given design threshold,
- the overall building costs exceed a predefined limit, and so forth.

If the envelope case is successfully repaired, then it is labeled as the *repaired building case.*

5.5. RECOMMENDED CASE

If the case testing was successful, then the CBR system ends up with a verified case, which is submitted as the *recommended building design solution*; otherwise, on failure, the current base case is eliminated from the list of suitable design alternatives, and the system restarts with the case transformation procedure. A successful design solution might be appended to the case library with the same indexes used for the retrieval of the base case.

Figure 5-4 outlines the entire data processing flow that occurs within the CBR framework.

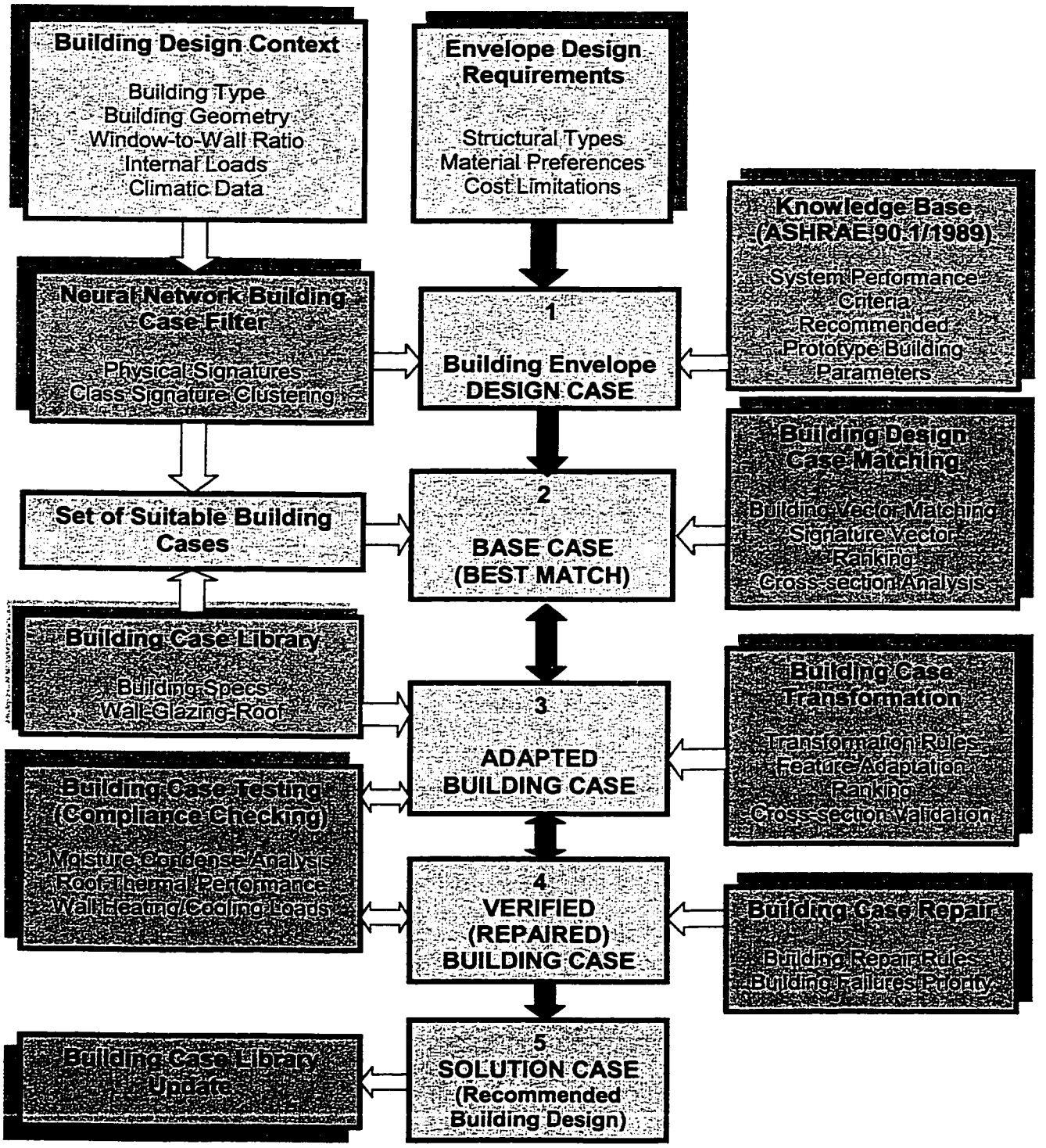


Figure 5-4. Envelope Design Flow in a CBR Framework

6. CBR MODEL FOR BUILDING ENVELOPE DESIGN

6.1. MODEL CHARACTERISTICS AND ASSUMPTIONS

The analysis of the building-envelope design process identified the following requirements for a computerized tool:

- Recover from design errors (while the design process passes through several iterations).
- Handle incomplete design specifications.
- Assess the design performance.

To build a CBR model for building-envelope design, the following assumptions were made:

- There is no uncertainty attached to the experiential knowledge embedded in previous design cases (or, in other words, all cases in the reference library are equally valid).
- ASHRAE 90.1 recommendations can be extended to cover Canadian locations as well (and not only US locations).
- The simplified simulation technique employed to assess the design performance will not favor any building design. In other words, by limiting the complexity of the calculations involved, the error associated with each performance assessment is within the same range of magnitude for all materials, components, and systems.

6.2. THE MODEL ARCHITECTURE

The proposed CBR model for building-envelope design includes the following components:

- *Envelope Data-Acquisition module.*
- *Building Case-Library.*

- *Envelope-Designer* module.
- *Performance-Analysis* module.

According to the design flow presented in Figure 5-4, the data acquisition part delivers the so-called "building-design case" that will be further matched against the cases stored in the reference database (i.e. the case library). Then, the CBR designer module will attempt to associate the design specifications (embedded into the design case) with a real building-envelope design case (the so-called "adapted building case"). Finally, the performance-analysis module will check if the proposed design conforms to the required performance criteria, and it will attempt to repair if below a given performance threshold. Figure 6-1 presents the interaction between the major parts of the proposed CBR model.

The technical aspects of the methodology proposed will be further clarified through a building-envelope design example starting with the following requirements:

- Building location: *Montreal, Quebec, Canada*
- Building-occupancy type: *office*
- Number of floors: *10*
- Total floor area: *25,600 m^2 (~275,556 ft^2)*
- Additional requirements: *Concrete* walls and roof, *double glazing* windows

Appendix C evaluates the preliminary performance of the building-envelope model that matches the above requirements while conforming to the Quebec Construction Law and to the ASHRAE Standard 90.1/1989. Based on the above design requirements, a design target, also known as *Design Case*, will be further defined.

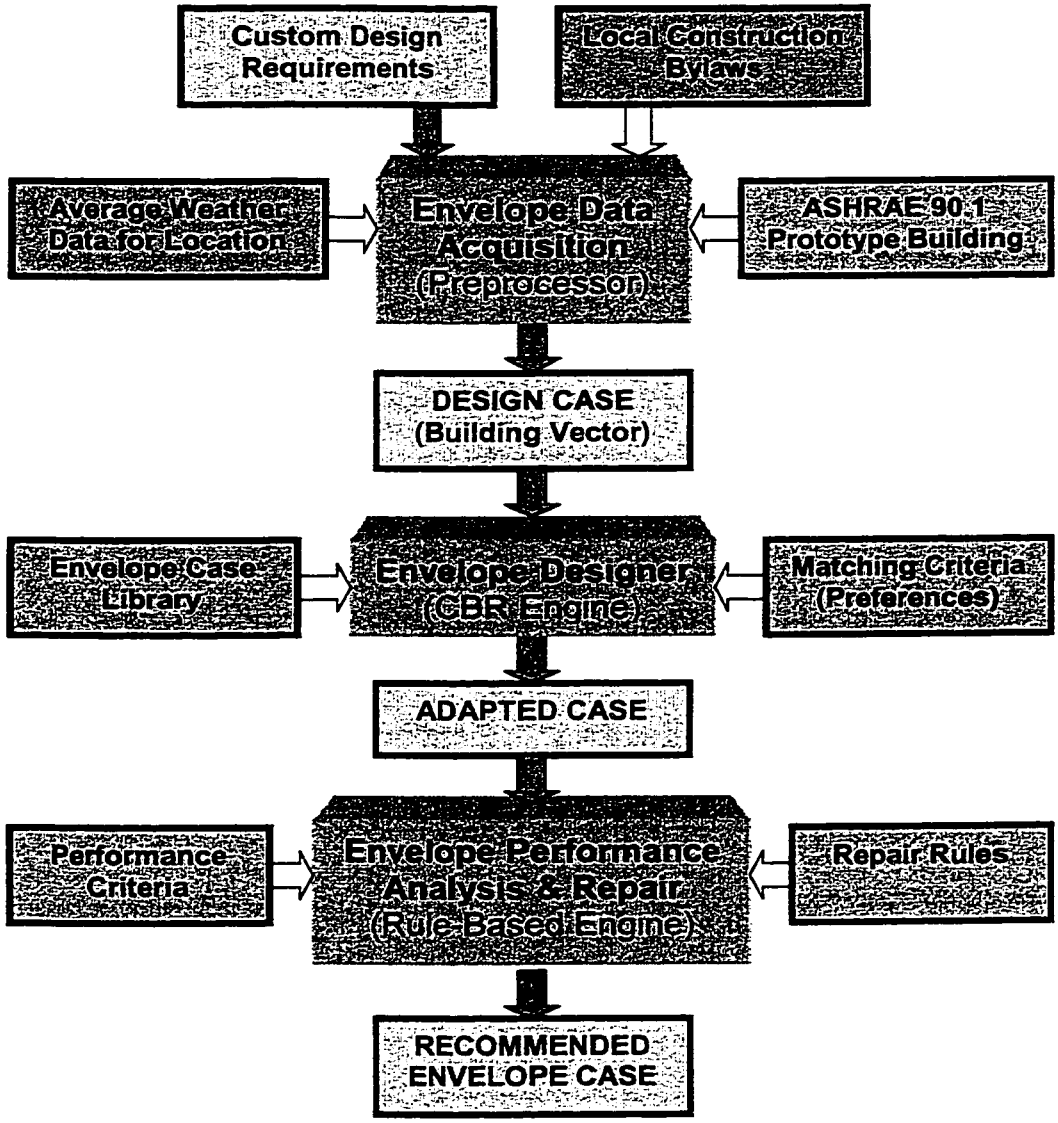


Figure 6-1. CBR Model for Building Envelope Design

6.3. ENVELOPE DATA ACQUISITION

The data-acquisition module is responsible for defining the design context under which the new building envelope is generated. It starts with a minimal set of user-specified requirements, and it ends with a full specification for the *Design Case*. The complete set of design specifications is established as a function of the input requirements, and a target design case is generated accordingly. The design case is assembled around the building vector to be matched by the new envelope.

The input data to be considered while designing a building envelope include:

- Average climatic data for the building location.
- Local regulations imposed by construction bylaws.
- Design requirements for the new building (including size, structural needs, aesthetic preferences, cost margins, and so forth).
- Performance recommendations (such as those in ASHRAE 90.1).

The climatic data for the target location is usually available from the local weather station or can be inferred from the data available for neighboring sites. The weather data for building-envelope design include the average temperatures and the humidity on the coldest and the warmest months. Appendix A details the types of climatic data employed for building-envelope performance analysis by the System Performance Criteria of ASHRAE Standard 90.1/1989, and Figure 6-2 shows the average weather data for Montreal, Quebec, Canada.

It is not unusual to find that the preliminary design requirements for a new building envelope are incomplete. Even some of the parameters included in the building vector might yet to be specified. In that case, the available domain knowledge could be used to infer the missing information. A very good source of information is the prototype building description generated

by the System Performance Criteria. For instance, the design example proposed in Section 6.2 lacks specifications such as the aspect ratio and the window-to-wall ratio. For the aspect ratio, the System Performance Criteria suggest a default value of 2.5. The window-to-wall ratio has recommended values not only in the System Performance Criteria, but in the local construction bylaws (such as Quebec Construction Law) as well.

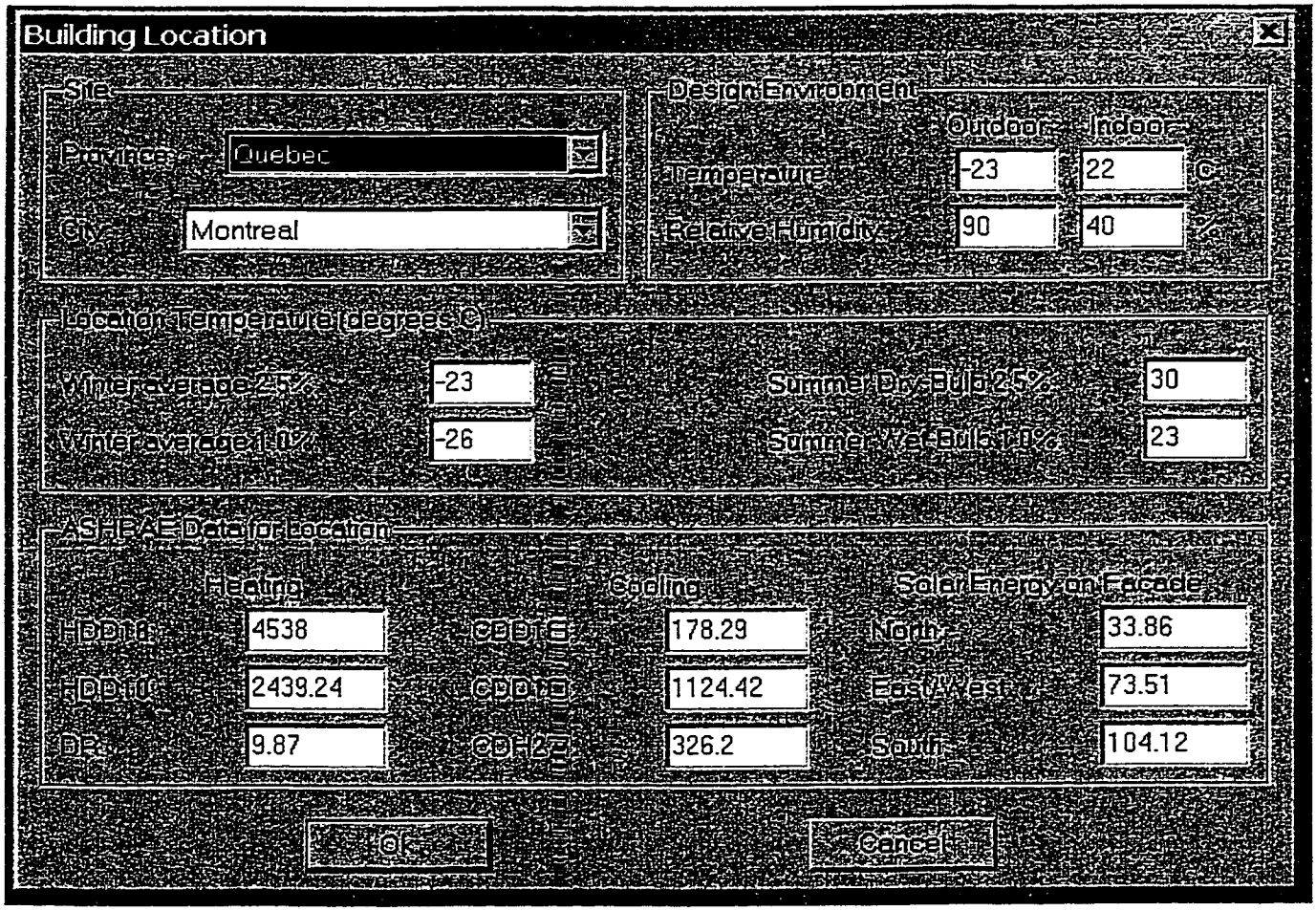


Figure 6-2. Average weather data for Montreal

The knowledge base includes three types of design information:

(a) *Building-geometry data* (such as building shape and orientation, average floor height, window-to-wall area, and so on);

(b) *Occupancy-related data* (such as number of persons per square meter, schedules of operations, average lighting, and equipment loads, to name just a few);

(c) *Energy-performance data* (such as recommended U-values for walls, glazing and roof, average heat generation per person, standard thermostat settings, average ventilation rates, and so forth).

Appendix C shows how the initial (and incomplete) design specifications can be augmented to provide all the information required for a preliminary assessment of the energy performance for the building envelope.

In summary, by employing the digitized categories defined in Section 4.2.2, the design example of Section 6.2 can be re-mapped into a design context as follows:

- Building location: *Montreal, Quebec, Canada*
- Building type category: *2 (office)*
- Building height category: *2 (medium-low rise: 4 to 10 stories)*
- Building size category: *7 (25,600 m^2 > 75,000 ft^2)*
- Aspect Ratio: *2.5*
- Window-to-Wall ratio: *0.25* (average value)
- Wall type: *Concrete*
- Fenestration type: *Double glazing*
- Roof type: *Concrete*

Accordingly, the new building-envelope design is to be associated with the following building vector:

$$Building\ Design\ Vector = \{2,\ 2,\ 7,\ 2.5,\ 0.25\}$$

This vector will be used as primary selection criteria while searching for building-envelope cases that match the design context.

Another part of the data acquisition process is to define the envelope-performance level to be reached. The System Performance Criteria of ASHRAE 90.1 offers a performance target based on a *Prototype Building*. The prototype building is assembled as a theoretical model of the real building currently designed by using the same general description parameters such as the building type and the building location. The prototype building description corresponding to the current *Design Case* is shown in Figure 6-3.

The objective here is to find the most suitable building envelope for the design requirements of Section 6.2 while reaching the performance level of the equivalent *Prototype Building*. The solution will be based on a library of validated envelope-design cases.

6.4. CASE-BASED ENVELOPE DESIGNER

The case-based designer starts with a *Design Case* and attempts to find the closest match in a library of known cases. Furthermore, the best match (also known as *Source Case*) is modified to fit the new design context and becomes the *Adapted Case*.

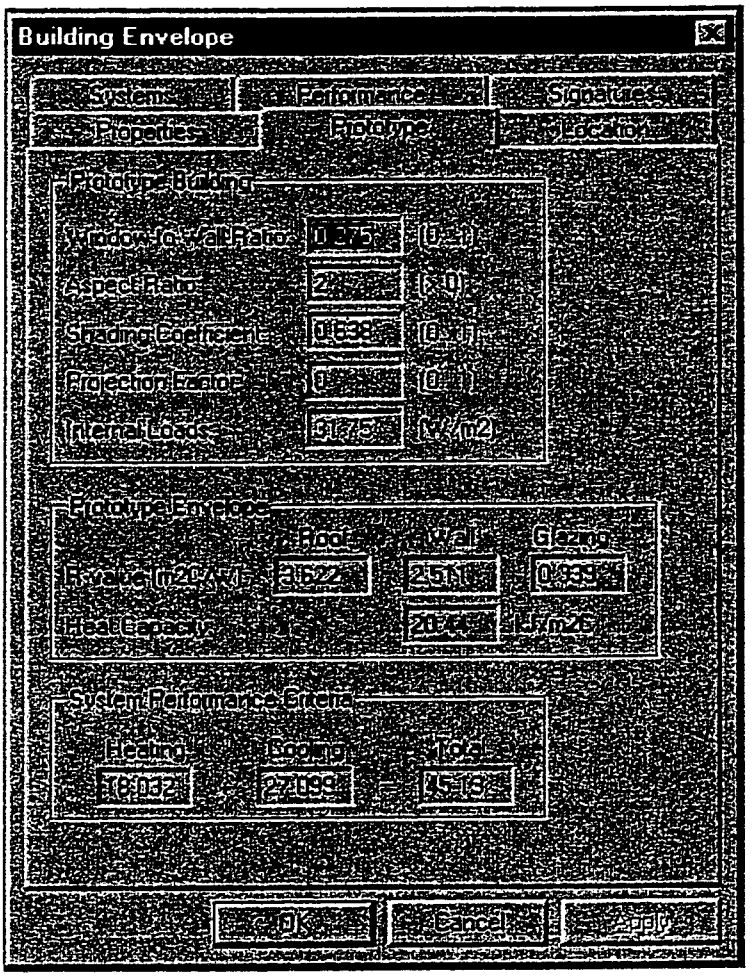


Figure 6-3. Prototype Building Parameters

Usually, the matching operation implies a set of math-intensive computations, therefore, it is advisable to narrow the matching to a subset of cases obtained through an intelligent filtering procedure. As described in Section 5.1, the filtering mechanism proposed here is anchored on the concept of a physical signature. The physical signature associated with a building-envelope case is a vector including up to 16 elements (i.e. there are 4 position and 4 value parameters defined for both the roof and the wall components of the envelope). Prior to any matching attempt, a

back-propagation artificial neural network (ANN) filter is trained to associate the building vector with the physical signature for all the cases in the library.

6.4.1. Matching the design case

The matching process has three steps that are triggered incrementally as follows:

I. Design building vector is used for a coarse filtering of library case.

II. Design physical signature (also known as *Class Signature*) is used to rank cases sharing the same building vector.

III. The highest-ranking are submitted to an extended evaluation procedure where the envelope composition is being considered as well.

The top-ranking cases, up to a pre-defined maximum number n (known as the *matching boundary threshold*), are assembled in the so-called set of *suitable design alternatives* (SDA) and compared with the Design Case. The comparison is performed through an extended evaluation function summing three terms as follows:

$$F_{D\to C} = F^{1}_{D\to C} + F^{2}_{D\to C} + F^{3}_{D\to C}$$

where the three terms represents:

$F^{1}_{D\to C}$: matching component associated with the building vector

$F^{2}_{D\to C}$: matching component associated with the physical signature vector

$F^{3}_{D\to C}$: component accounting for layer type and material-class matching

The first two terms ($F^{1}_{D\to C}$ and $F^{2}_{D\to C}$ respectively) are the weighted Euclidean distances from the *Design Case* (D) to the library case under scrutiny (C) computed with respect to the building vector and physical signature vector as shown below:

$$F^{k}{}_{D\to C} = \sum_{i} w_{ki}\sqrt{\left|D_{ki}^{2} - C_{ki}^{2}\right|}$$

where w_{ki} is the weight associated with the component i of vector k.

The last term ($F^{3}{}_{D\to C}$) accounts for layer class and type matching through the following formulae:

$$F^{3}{}_{D\to C} = \sum_{i} (w_{i3}L_i + w_{i4}M_i)$$

where w_{i3} and w_{i4} are the weights associated with the layer i for the layer type (Boolean parameter L) and the material class (Boolean parameter M) respectively. For example, if the *Design Case* includes a requirement for a concrete wall, and the library case under scrutiny has as structural layer a concrete block, then both parameters L ((*Structure* layer is present) and M (*CONCRETE* type matched) are 1. If, however, the library case has the structural layer specified as steel studs, then L is 1 (*Structure* layer is present) but M is 0 for the layer type (*CONCRETE* type not matched). Finally, if the library case includes a curtain wall type, then L is 0 (*Structure* layer is not present).

Note that the matching score yielded by the evaluation function is highly dependent on the weighting coefficients w. The proper selection of those coefficients has a major impact on the CBR outcome and it should be based on expert analysis.

Not all the building characteristics assembled into the building vector carry the same weight. Normally, the building occupancy type and aspect ratio are less important than the building height classification and/or the window-to-wall ratio. The building designer can alter the weight associated with each feature to direct the similarity assessment according to his/her preference. That is one of the major benefits of employing a CBR methodology: the capability to shift the design focus from one feature to another by fine-tuning the retrieval process.

This research identifies the following guidelines for selecting the weighting coefficients:

- *Coefficients associated with the building vector should significantly outweigh those associated with the physical signature vector.* The physical signature vector is used to facilitate further ranking among cases sharing the same building vector.
- *The values of the weighting coefficients for the building vector parameters should decrease according to following scale: 1) building height and building size, 2) building occupancy type, 3) window-to-wall ratio, and 4) building aspect ratio.* Since the building dimensions (height and total floor area) have the most noticeable impact on selecting the envelope structure, they should be associated with the biggest values for weighting coefficients. Window-to-wall ratio is a very important parameter as well, but its recommended value is usually compromised vis-à-vis the building occupancy type and the custom preferences.
- *Weighting coefficients associated with layers more difficult to replace (such as the structural layer) or with layers defining the envelope type (such as the air cavity and/or the cladding) should have larger values than other layers.*

As an example, the above evaluation function was employed to retrieve the best match for the Design Case proposed in Section 6.2 from a set of approximately 25 building-envelope cases.

The top three cases in the list of suitable design alternatives are shown in Tables 6-1, 6-2, and 6-3. Note that the only case fully matching the design building vector is ranked third. Moreover, a case classified as *Residential* is ranked above a case classified as *Office* (the building type required in design specifications).

Table 6-1. Top ranked case in the list of suitable alternatives

#1	Building Vector	Wall	Glazing	Roof
Out		Brick		Stone ballast
	Office	Air space		Three-ply build-up roof
	Medium-low rise	Glass fiberboard rigid		Phenolic foamboard
	Size type 6	Concrete block	Double glazing 12.7 mm	Glass fiber sheathing
		Polyethylene sheet		Steel deck
		Furring		Gypsum board
In		Gypsum board		Acrylic paint

Table 6-2. Case ranked second in the list of suitable alternatives

#2	Building Vector	Wall	Glazing	Roof
Out		Stucco		Stone ballast
	Residential	Plain cement panel		Geotextile membrane
	Medium-low rise	Air space		Polystyrene rigid
	Size type 5	Glass fiber	Double glazing 12.7 mm	Polyethylene sheet
		Polyethylene sheet		Roofing membrane
		Concrete block		Concrete slab
		Furring		Plaster on gypsum
In		Gypsum board		Acrylic paint

Table 6-3. Case ranked third in the list of suitable alternatives

#3	Building Vector	Wall	Glazing	Roof
Out		Brick		Roofing membrane
	Office	Air space		Glass fiber rigid
	Medium-low rise	Glass fiber rigid		Polyethylene sheet
	Size type 7	Exterior sheathing	Double glazing 12.7 mm	Concrete slab
		Steel studs		Parging Finish
		Batt insulation		Acrylic paint
		Polyethylene sheet		
		Furring		
In		Gypsum board		

The explanation of that ranking is linked to the cross-section analysis. While the case ranked second has a concrete wall, the case ranked third has a steel stud wall. *Concrete* is a keyword identified among the additional input requirements for the wall design (at the beginning of Section 6), and is a structural qualifier that is usually associated with the structural layer. Since a structural layer is more difficult to replace, more weight was allocated for matching it (as a feature difficult to adapt) than for the type of building and/or the physical signature.

6.4.2. Adapting the matched case

The best-match case is modified towards the design requirement through an iterative process as described below:

I. The layers that do not match the custom requirements are marked for replacement.

II. The case library is searched to identify potential replacement layers.

III. Replacement alternatives are ranked according to the level of similarity between the envelope component containing them and the envelope component to modify.

IV. The highest-ranking alternatives are used to replace the unmatched layers.

V. A cross-section analysis is performed to validate the new envelope.

VI. If the cross-section analysis failed, then the entire envelope component (that is, wall or roof) is marked for replacement.

The outcome of a successful case transformation is the *Adapted Case.* The new envelope is then submitted to a set of compliance checking tests before being labeled as the *Recommended Solution Case.*

6.5. ENVELOPE COMPLIANCE ANALYSIS

There are two types of tests performed for evaluating the envelope performance:

1. *Validation tests* (checking that the envelope case under scrutiny is a feasible design alternative), and
2. *Performance tests* (qualifying if the envelope case reached the desired level of performance).

The validation tests are performed first (because they can discard the envelope as a design alternative), and include:

1. Checking that there is no water vapor condensation within any envelope component and that there is no surface condensation as well.
2. Verifying that the new envelope case has some of the overall parameters such as cost and thickness within prescribed limits.
3. Assessing that the envelope case under scrutiny is a valid construction alternative.

The performance simulation routines adopted by this research are based on the System Performance Criteria of ASHRAE Standard 90.1/1989 (ASHRAE, 1989). The performance tests are linked to the energy conservation and focus on the overall thermal resistance of the envelope:

1. The overall R-value of the roof is below the ASHARE recommended value.
2. The sum of total heating and cooling loads for the wall is lower than the value computed for the prototype building.

6.6. REPAIR STRATEGY

An adapted envelope case is considered as a *FAILURE* if any of the following two conditions is met:

1. There is concealed or surface moisture condensation.
2. The thermal performance of the envelope does not reach the level recommended by ASHRAE Standard 90.1.

The attempts to repair a failed envelope case will focus on the overall vapor and thermal resistance of the envelope component that exhibits the problems. The algorithm to repair an envelope case is the following:

I. Evaluate the moisture performance by computing the water vapor flow through the envelope component, *VF* (not including the concealed condensation). The heat and water vapor flows are computed based on the equations of Appendix E.

II. If there is no moisture condensation, then go to step (V).

III. Compute the required overall vapor resistance, R_v , with the following formula:

$$R_v = |P_{in} - P_{out}|/VF$$

where water vapor pressure P is obtained by multiplying the relative humidity with the water vapor pressure at saturation for the given inner or outer temperature:

$$P_i = RH_i * P_{sat}(T_i)$$

IV. If the vapor barrier resistance is less then R_v, then attempt to increase the vapor barrier layer. If that is not possible (already too thick), then attempt to move the vapor barrier toward the warmer side of the envelope. If the vapor barrier cannot be further moved, then go to step (XI), otherwise, go to step (II).

V. Evaluate the thermal performance by applying the System Performance Criteria of ASHRAE 90.1.

VI. If the envelope meets the criteria, then go to step (X).

VII. If the overall thermal resistance of the failed envelope component is below the equivalent value recommended for the prototype building, then modify the insulation layer accordingly.

VIII. If the insulation-layer thickness exceeds a pre-defined threshold (say 89mm), then split the insulation layer into two parts as suggested by the practical considerations of Section 2.2.

IX. If the insulation layer cannot be modified anymore, then try to adjust the window-to-wall ratio. If that is not possible, then go to step (XI), otherwise go to step (V).

X. Declare the envelope case *REPAIRED* and stop.

XI. Discard the envelope case as *DESIGN FAILURE* and stop.

7. MODEL IMPLEMENTATION

The proposed model has been implemented in a prototype system known as CRED (Case Reasoner for Envelope Design) aimed at providing computerized assistance in the preliminary stage of the building-envelope design process. It includes three databases as follows:

- a database recording the average weather information for target building sites as required by the System Performance Criteria of ASHRAE Standard 90.1/1989 (*Site DB*),
- a database of construction materials recording the values of physical parameters such as thickness, density, thermal and vapor resistance, specific heat, and emittance (*Material DB*), and
- a database of verified building-envelope design cases (*Case DB*).

The reasoning core of the CRED system consists of three AI modules as follows:

- an artificial neural network (ANN) filter used for ranking alternatives during the matching phase,
- a case-based reasoning (CBR) engine used not only for identifying the most suitable design alternative (that is, matching), but for case adaptation as well, and
- a rule-based engine implemented by an automatic reasoning module (ARM) and used to process the domain knowledge rules employed for case transformation and repair.

Moreover, CRED includes a reporting utility capable of generating, displaying and printing a full report regarding the building-envelope case currently under scrutiny.

The architecture of CRED is shown in Figure 7-1. All the components of CRED were completely implemented without any support from a commercial of-the-shelf software program.

Developing from scratch all major software pieces (that is, ANN, CBR and KBES) insured the flexibility required to integrate them seamlessly.

The building-envelope design process starts with an *input set* that defines the design context and includes all the envelope requirements, and ends with the *design outcome* that is either a proposed building envelope or a failed attempt to generate one.

7.1. COMPUTING ENVIRONMENT

The system is implemented using object-oriented design principles and is written in C++ for the Win32 environment (Windows 9x or Windows NT/2000). The C++ classes are built on top of MFC (Microsoft Foundation Classes) and the message-passing mechanism is following the COM (Component Object Model) architecture. The hardware requirements are as follows: Pentium-based personal computers with 32 MB of RAM.

The CRED implementation includes an approximate number of 30,000 lines of C++ code. Appendix H details the main constants and the class templates, while Table 7-1 lists the major components and their respective code sizes.

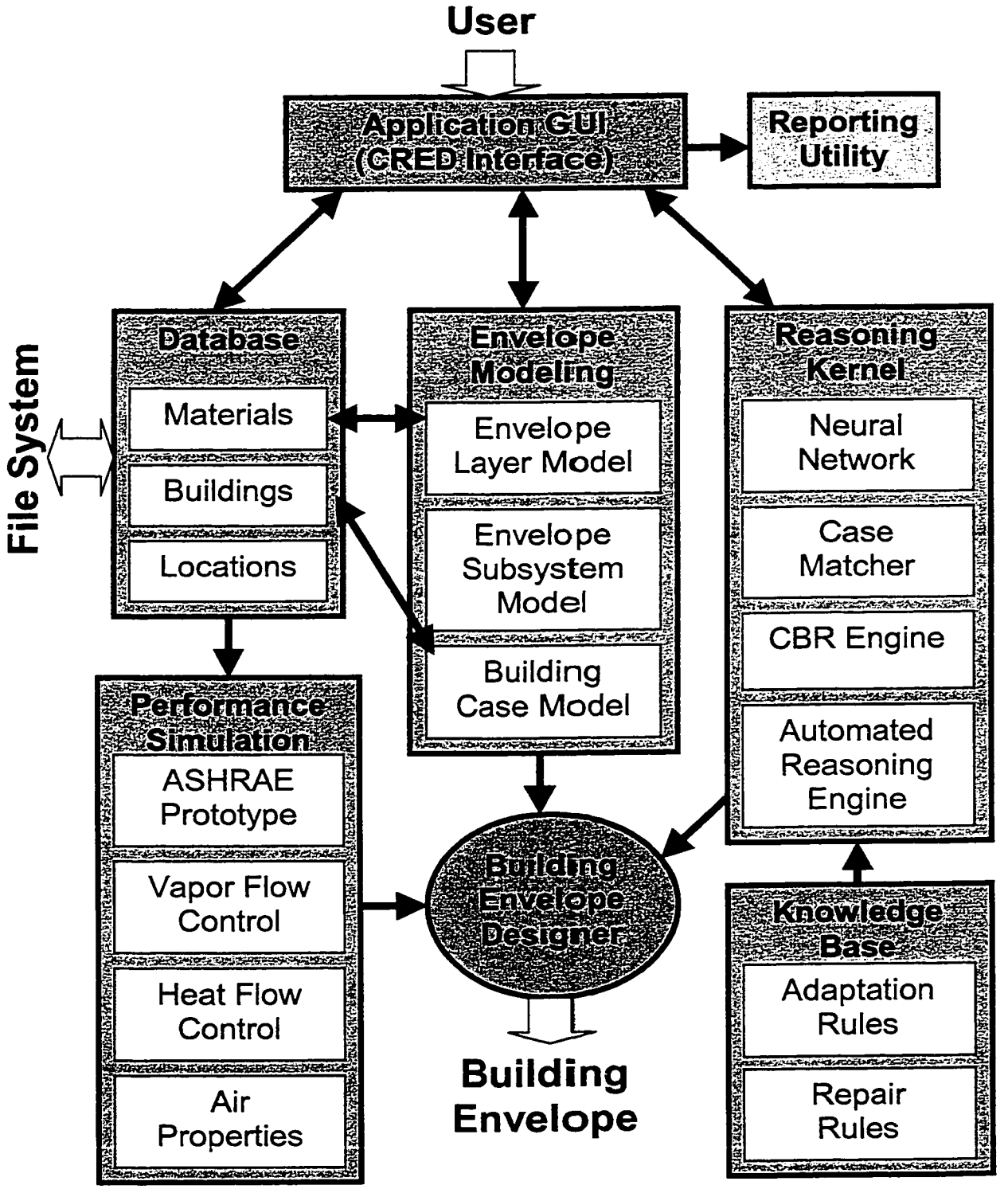


Figure 7-1. CRED Architecture

Table 7-1. CRED code size

Module	Description	Code Size (No of C++ lines)
User interface	Win32 message handling Main application window Setup dialogs and property pages	5,700
Reporting utility	Report generation and viewer Print utility	3,300
Common utilities	Global operations Tracing and debugging utility	2,000
Case-based reasoning kernel	Reasoning flow control Case matching control Case adaptation Learning and updating module	3,200
Case matcher	Physical-signature filter Ranking modules Matching module	1,100
Artificial neural network	ANN model Training utility	2,500
Automated reasoning module	Rules loader Horn clauses processor Resolution strategies	5,200
Building envelope modeling	Building case Layer & System model Envelope verification module Case database model	2,800
ASHRAE System Performance Criteria	Prototype building model SPC recommended values Cooling & Heating criteria	1,200
Building envelope performance evaluation	Moisture migration equations Air-cavity calculations Air-film calculations	800
Material Database	Physical properties evaluation Database access mechanism Search engine	1,400
Site Database	Weather data encoding Database access mechanism	500
CRED		**29,700**

7.2. MODEL COMPONENTS

CRED includes the following major components:

- User interface (Win32 GUI),
- Reporting utility,
- Envelope case library (*Case DB*),
- Database of weather-related information for all major Canadian locations (*Site DB*),
- Database of common building materials (*Material DB*),
- Building-envelope knowledge base of adaptation, checking and repair rules,
- Performance simulation module (including the procedures for calculating the heat and vapor flow through the envelope, and the implementation of the System Performance Criteria of ASHRAE),
- Artificial neural network,
- CBR-based case matcher,
- Case adaptation and validation module.
- Automated reasoning engine,
- Building envelope designer (controlling the reasoning flow).

For case adaptation and envelope validation, the CBR engine is enhanced with a rule-based engine. That rule-based engine is actually an automated reasoning module (ARM) that infers facts through resolution. It is implemented to support both forward and backward chaining to include predicate logic techniques, such as demodulation and paramodulation, and various strategies as detailed in Appendix G.

7.3. DESIGN PREREQUISITES

Before attempting to use CRED for building-envelope design, the case library should be filled with a comprehensive collection of successful building-envelope design cases. Then, the ANN filter should be trained to match the building vectors of those cases with their physical signatures computed for a target-building location. The training algorithm and its associated heuristics are detailed in Appendix F.

7.3.1. Training the ANN case filter

The artificial neural network employed by CRED for filtering the cases is a back-propagation feed-forward layered network. The ANN yields a more reliable outcome if trained on a larger number of significant (i.e. diversified) samples. Training the network involves the following steps:

- Selecting the network architecture (meaning selecting the number of cells in the middle layer since the input and output layers are usually imposed).
- Selecting the training constants such as the *learning rate* (also known as *beta* constant), the *momentum* (or *alpha*) constant, the *noise factor*, the type of training (case or epoch updating), and the *error threshold* used as a stopping condition.

Figure 7-2 shows the default values of the training constants, and Appendix F describes in more details the algorithm and the heuristics applied for training the neural network. Following extensive training and testing experiments, the optimal layout for the neural network filter was found to include two middle layers of five and respectively four cells each as shown in Figure 7-3.

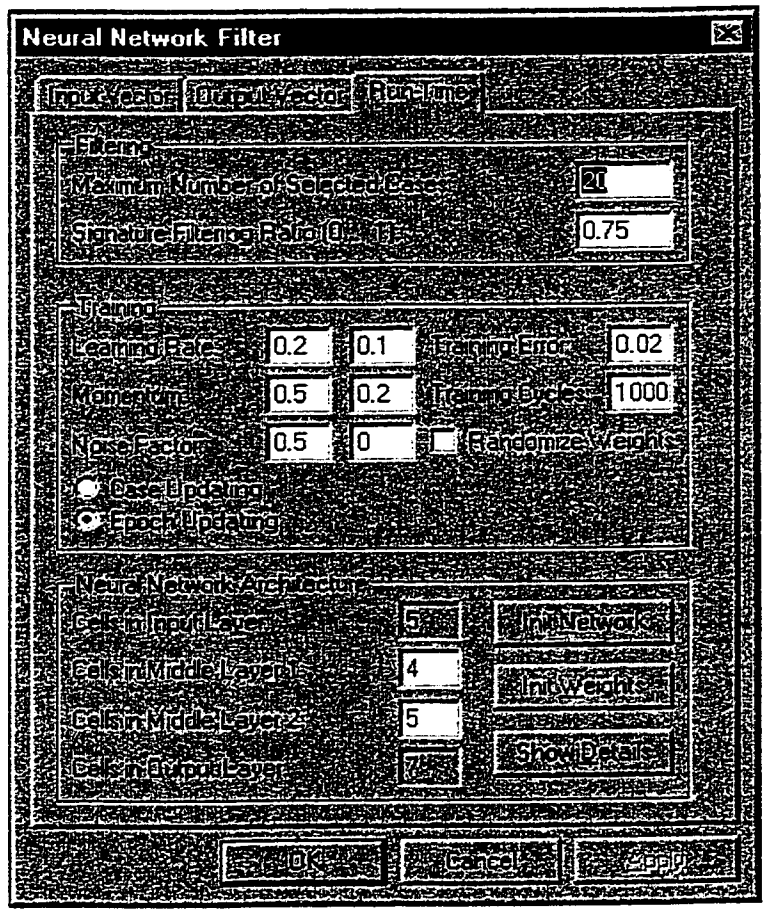


Figure 7-2. Neural Network Training Set-up

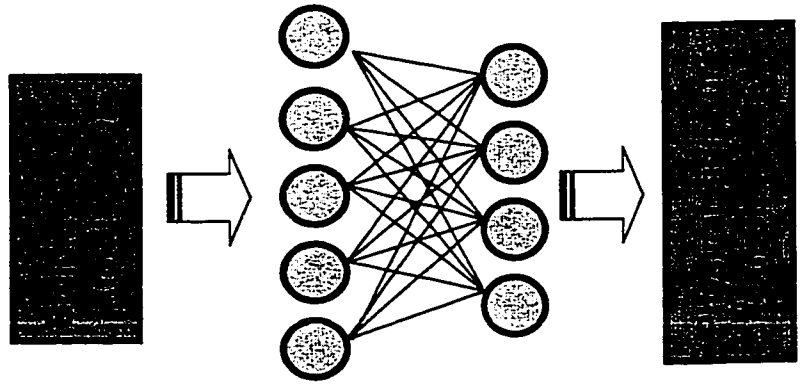


Figure 7-3. Neural Network Layout

7.3.2. Setting up the design case

I. Select a building location from the *Site DB*, and collect the necessary weather related data (Figure 6-2).

II. Establish the salient features of the new building design:

- *building type* (intended usage of the new building)
- *building height category* (total number of floors)
- *building size category* (total floor area)
- *building aspect ratio* (building width over building depth ratio)
- *window-to-wall ratio* (percentage of glazing area of the total wall area)

III. Interpolate missing information and secondary features from the associate knowledge base (based on the data extracted from ASHRAE 90.1/1989):

- *building occupancy* (the average number of persons per unit of floor area),
- *internal loads* (general electrical service loads including lights and equipment),
- *shading coefficient*,
- *projection factor* (external horizontal shading projection depth).

IV. Include construction preferences such as structural type, interior and/or exterior finish, roof pitch, etc.

V. Assemble the *Design Requirements* of the new design case

For instance, the design-case setup for the design example of Section 6-2 is shown in Figure 7-4. The shading coefficient, the projection factor, and the internal loads were inferred based on the System Performance Criteria of ASHRAE 90.1/1989.

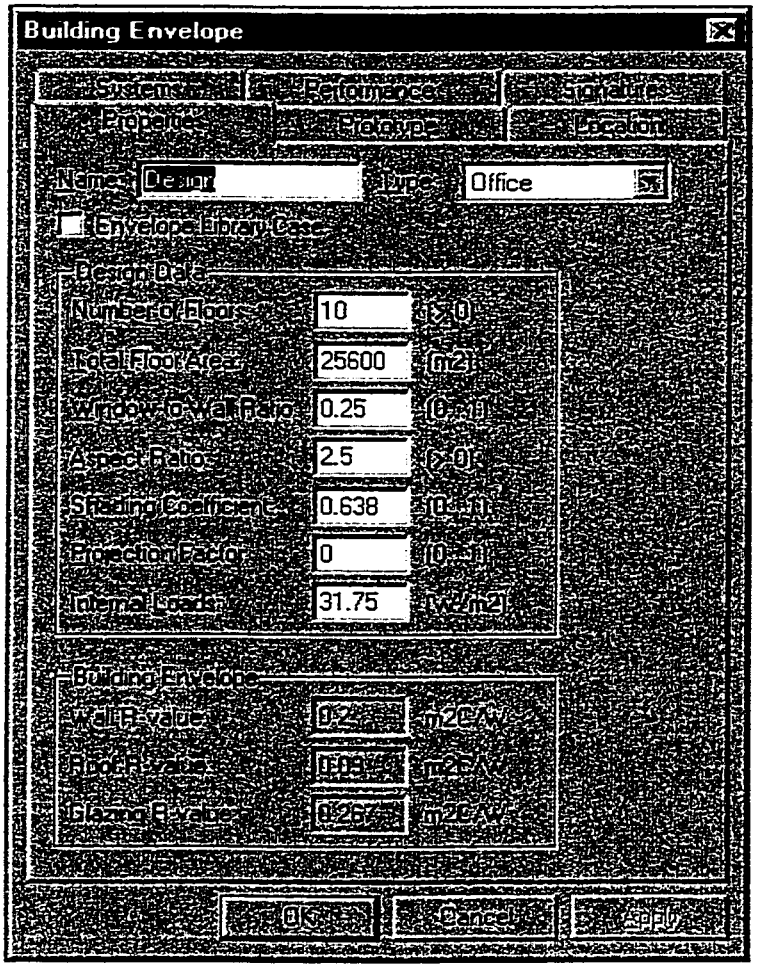


Figure 7-4. Design Case Set-up

7.4. DESIGN CASE PRE-PROCESSING

Prior to "firing" the case-based reasoning process, the design context should be checked for validity. For instance, if the wooden-wall type is required for a building with more than three stories, then there is a reasoning conflict since such an envelope is not a practical solution.

The set of verification rules fed into the rule-based engine includes:

IF (Number of floors <= 3) **THEN** *Curtain Wall* is not allowed.

IF (Number of floors > 3) **THEN** *Wooden Wall* is not allowed.

IF (Number of floors > 10) **THEN** *Simple Glazing* is not an option.

IF (Wall is not of Framed Type) **THEN** *Batts Insulation* is not an option.

IF (Cladding and Structure cannot be paired) **THEN** *Cladding* is to be replaced.

IF (Wall and Roof cannot be paired) **THEN** *Roof* is to be replaced.

The design case pre-processing includes the following steps:

- Check the design context for validity.
- Discard and/or replace conflicting specifications.
- Assemble the prototype building (as specified by System Performance Criteria of ASHRAE), and establish the design requirements to be achieved (shown in Figure 6-3)
- Retrieve the *Design Signature* by feeding the building vector of the design case into the ANN filter.
- Adjust the matching metrics and establish the similarity metrics for matching (that is, assemble the list of keywords to be used for cross-section analysis).
- Assemble the *Design Case Context.*

7.5. CASE MATCHING

- Initialize the search-engine.

- Establish the preliminary matching-threshold error (that is, the acceptable error at building vector level).
- Sort the existing *Building Case Library*, according to current design specs, to establish the list of building cases suitable to current design context (within the above error threshold).
- Rank the list of suitable alternatives in the descending order of their Euclidean distance to the design requirements (distance computed based on the general building characteristics included in the building vector and the physical signature).
- Establish the maximum number of building cases to be further considered for matching (that is, select a filter threshold).
- For all the retained buildings, calculate the matching score according to the established similarity metrics (i.e. matching polynomial function with adjustable coefficients).
- Rank building cases according to their matching score.
- Label the top case as *Best Match Case.*

The data structures and the C++ class templates used by the CBR engine are detailed in Appendix H. Appendix J shows a sample trace of the execution steps in CRED required for the designing of a new building envelope.

8. CONCLUSIONS AND RECOMMENDATIONS

8.1. SUMMARY

This thesis presents an alternative approach for automating the building-envelope design process by using the case-based reasoning methodology. A conceptual model is proposed for employing the experience embedded in known building-envelope cases while designing a new building envelope. The steps to be taken in this case-based framework for building-envelope design are summarized below:

- *Previous design experience is stored in building-envelope cases by limiting the description to avoid unnecessary encoding complexity.*
- *A comprehensive case library includes cases covering most of the possible design alternatives.* Since the building-envelope design outcome depends upon the size and quality of the case library, the information embedded in an envelope case has to be verified and validated before adding that case to the library.
- *The CBR model is equipped with capabilities to rapidly identify the library case that best matches the current design requirements.* The matching process is computationally intensive and it is limited to a subset of suitable cases by using the physical signature to further rank cases sharing the same building vector.
- *The matching mechanism is enhanced with a cross-section analysis component.* The proposed CBR model facilitates the comparison between two envelope cases by considering both the layer type matching and the material class matching.
- *The case adaptation has to be verified through a validation mechanism.* A newly generated building-envelope case is usually the result of modifying an existing case along

certain layers. A library-wide search is performed to ascertain if the newly introduced or modified layers can coexist within their current neighborhood.

- *The adapted case is checked against the performance requirements through a simulation procedure.* The test procedure checks that the adapted case has no surface or concealed moisture condensation and that its energy performance level conforms to the recommendations of the System Performance Criteria of ASHRAE 90.1/1989.
- *An adapted case failing the performance tests is submitted to rule-based repair engine.* The case-repair mechanism includes rules of how to modify the overall vapor and/or thermal resistance of the envelope until the performance requirements are met.

8.2. CONCLUSIONS

The major conclusions of the present investigation are the following:

- *Learning from past experiences through a computerized framework is beneficial in the preliminary stage of the building-envelope design process.* The envelope designer has an opportunity to explore and analyze various design alternatives before committing to an envelope design detail.
- *The complexities in the building-envelope design process recommend a computer-driven design process.* If the quest for energy efficiency is to meet the performance level recommended by the ASHARE 90.1 Standard, then the amount of computation required is staggering (for an unassisted designer using manual calculations). In addition to a large number of performance attributes to be considered, the selection of building materials is subject to conflicting performance requirements depending on the climatic conditions, construction regulations, and custom objectives. The analysis

of various design alternatives (as defined by changing the construction materials) implies a large amount of calculations to re-assess the overall performance of the envelope.

- *There is a need for a knowledge-driven mechanism to infer missing information when the building-envelope design specifications are incomplete.* Building-envelope design decisions are often made prior to the preparation of the schematic drawings. At that preliminary stage, it is critical for the designer to investigate several design alternatives to eventually choose an optimal solution. Missing design specifications, however, hamper the analysis of alternatives. Domain knowledge, construction bylaws, and renowned standards (such as ASHRAE 90.1) can be employed by a computerized reasoner to fill the information holes in incomplete design specifications. For instance, CRED infers missing data from the parameters defined for the prototype building in the System Performance Criteria of ASHRAE 90.1.
- *Economizing knowledge representation is required to keep the case-based reasoning process within reasonable limits of computational time.* Alternative mechanisms, such as the lexical parsing associated with the cross-section analysis, might be used to compensate for the lack of case description details.
- *Computer- generated designs should be validated through simulation and eventually repaired if they fail to deliver the expected performance.* Testing and repair are an integral part of the case-based reasoning framework. System Performance Criteria of ASHRAE Standard 90.1 offered a useful methodology for integrating the performance-compliance checking within a computerized system.

8.3. CONTRIBUTIONS

The present study improves the quality of future building-envelope designs by incorporating the heuristic knowledge from past successful designs and enabling designers to exploit computationally-intensive performance checking during the conceptual design stage. The main achievement of this research is the development of a systematic CBR framework for modeling the decision-making during the building-envelope design process as detailed below.

- *Comprehensive review of the research work related to applying case-based reasoning for building engineering design.* The review includes a detailed analysis of previous approaches as well as a comparative assessment of the advantages and disadvantages associated with the case-based reasoning approach.
- *Detailed analysis of the building-envelope design process with emphasis on the techniques to automate and computerize the process.* The analysis focused on the main problems to be solved, the incompatibilities and the complexity of the information faced by the designer.
- *Successful development of a knowledge-representation model for building-envelope objects.* The proposed model is built around the hierarchy of building-envelope objects facilitating problem decomposition at the component level. The knowledge-representation model was geared toward an economy of encapsulation and focused on information transfer from one envelope object to another.
- *Development of a complete CBR framework for automating the building-envelope design process.*
- *Resolving all major problems encountered when adopting CBR methodology for the building-envelope design domain.* The present thesis achieved the following goals:

- Expanding the classification criteria to facilitate retrieval by introducing the concept of physical signature.
- Mixing several techniques to optimize the results: artificial neural networks to filter the building-envelope cases, case-based reasoning to find the best-match case, and rule-based inference for repairing failed cases.
- Improving the matching metrics through cross-sectional analysis and proprietary heuristics (such as lexical parsing) without adding unnecessary encoding complexity.
- Increase the quality of the case adaptation by employing the case library for component adjacency validation.
- Checking the computerized design through a set of simulation and performance tests based on ASHRAE Standard 90.1.

❑ *Validation of the developed model through a software implementation.* The CRED prototype implements the CBR framework for building-envelope design using the current object-oriented programming techniques and user-interface technologies. As a software tool, CRED demonstrates that the objective of the research—provide computerized expertise for building-envelope design assistance—is indeed achievable.

8.4. RECOMMENDATIONS FOR FURTHER RESEARCH

This study models the envelope as an independent entity (without conceptual links to the rest of the building). Further research efforts can build upon this work to address other aspects of the computer-aided building-design process.

- *Provide a conceptual model for the interaction between the envelope and the rest of the building that integrates the envelope design within the building design process.*
- *Link the HVAC system and the selection of other building systems to the building-envelope design.*
- *Improve the data acquisition mechanism (to support CAD based descriptions) and expand the case library.*
- *Enable the model to support distributed architectures (to prepare for an eventual Internet-based implementation).*
- *Validate and improve the model by collecting input from experienced practitioners.*

This thesis is a step in the direction of computerizing the building design. The thesis does not dispute the impact of human creativity during the design process, it is clear now that the future will bring more and more powerful software tools for assisting building engineers and architects to generate consistently high-quality designs. The present thesis can be considered as a part of this effort to fully automate the building design.

REFERENCES

R1. BUILDING SCIENCE

ARNOLD, J. N., CATTON, I., EDWARDS, D.K., (1975), "Experimental Investigation of Natural Convection in Inclined Rectangular Regions of Differing Aspect Ratio", *ASME Paper* 75-HT-62, 1975.

ASHRAE, (1989), *Handbook of Fundamentals*, American Society of Heating, Refrigerating and Air-Conditioning Engineers, 1989

AYYASWAMY, P.S., CATTON, I., (1973), "The Boundary Layer Regime for Natural Convection in a Differentially Heated Tilted Rectangular Cavity", *International Journal of Heat Transfer*, volume 95, p.543, 1973.

CHURCHILL, S. W., CHU, H. H. S., (1975), "Correlating Equations for Laminar and Turbulent Free Convection from a Vertical Plate", *International Journal of Heat Mass Transfer*, volume 18, p.1323, 1975.

FUJII, T., IMURA, H., (1972), "Natural Convection Heat Transfer from a Plate with Arbitrary Inclination", *International Journal of Heat Mass Transfer*, volume 15, p.755, 1972.

HOLLANDS, K.G.T., UNNY, S.E., RAITHBY, G.D., KONICEK, L., (1976), "Free Convective Heat Transfer Across Inclined Air Layers", *International Journal of Heat Transfer*, volume 98, p.189, 1976.

HOLMAN, J.P., (1981), *Heat Transfer*, Fifth Edition, chapter 7: Natural Convection Systems, McGraw-Hill, 1981.

HUTCHEON, N.B., HANDEGORD, G.O.P., (1989), *Building Science for a Cold Climate*, Published by The National Research Council of Canada, 1989

INCROPERA, F.P., DeWITT, D.P., (1981), *Fundamentals of Heat Transfer*, chapter 9: Free Convection, John Wiley, 1981.

ROHSENOW, W. M., HARTNETT, J. P., (1973), *Handbook of Heat Transfer*, chapter 2: Thermophysical Properties, page 2-80, McGraw-Hill, 1973.

VINES, R. G., (1960), "Measurement of Thermal Conductivity of Gases at High Temperature", *Journal of Heat Transfer*, Transactions ASME, 82:48, 1960.

WEXLER, A., HYLAND, R. W., (1980), "A Formulation for Thermodynamic Properties of the Saturated Pure Ordinary Water Substance from 173.15 to 473.15 degrees Kelvin", Final Report, *ASHRAE Research Project RP216*, 1980.

R2. BUILDING ENVELOPE DESIGN

AMBROSE, J., (1991), *Building Construction – Enclosure Systems*, Van Nostrand Reinhold, New York, 1991.

ASHRAE, (1989), "Energy Efficient Design of New Buildings Except New Low-Rise Residential Building", *Standard 90.1/1989*, American Society of Heating, Refrigerating and Air-Conditioning Engineers, 1989.

BRAND, R., (1990), *Architectural Details for Insulated Buildings*, Van Nostrand Reinhold, New York, 1990.

CANADA MORTGAGE AND HOUSING CORPORATION, (1982), *Energy Efficient Housing Construction*, 1982.

FAZIO, P., BEDARD, C., GOWRI, K., (1989), "Knowledge-Based Approach to Building Envelope Design", *Computer-Aided Design*, volume 8, number 8, 1989.

FIBERGLASS CANADA CORPORATION, (1990), *Guide to Building Envelope Design*, 1990.

GOWRI, K., (1990), "Knowledge-Based System Approach to Building Envelope Design", *Ph.D. Thesis*, Centre for Building Studies, Concordia University, Montreal, 1990.

GOWRI, K., ILIESCU, S., FAZIO, P., (1992), "Case-Based Reasoning in Building Envelope Design", *Proceedings of the Seventh International Conference on Applications of Artificial Intelligence in Engineering (AIENG' 92)*, July 1992, Waterloo, Ontario, co-published by Computational Mechanics Publications and Elsevier Applied Science, pp. 1221-1238.

GOWRI, K., ILIESCU, S., FAZIO, P., (1995), "A Case-Based Model for Building Envelope Design Assistance", *Proceedings of the Second Congress on Computing in Civil Engineering*, June 5-8, 1995, Atlanta, Georgia, published by American Society of Civil Engineers, vol.2, pp. 171-178

ILIESCU, S., GOWRI. K, FAZIO, P., (2000), "Similarity Assessment in a Case-Based Reasoning Framework for Building Envelope Design", *International Conference on Construction Information Technology 2000 (INCITE 2000)*, January 17-18, 2000, Hong Kong, pp. 697-710.

ILIESCU, S., FAZIO, P., GOWRI. K., (1998), "A Case-Based Reasoning Framework for Preliminary Building Envelope Design", *First International Conference on New Information Technologies for Decision Making in Civil Engineering*, October 11-13, 1998, Montreal, Quebec, Canada, pp. 401-412.

JOHNSON, G.F., (1991), "Alberta Building Envelope Failure Analysis", *Technical Report from Alberta Municipal Affairs Housing Division*, June 1991.

LATA, J. K., (1973), Walls, "Windows and Roofs for the Canadian Climate", National Research Council of Canada, October 1973.

MASONRY COUNCIL OF CANADA, (1982), Guide to Energy Efficiency in Masonry and Concrete Buildings, 1982.

POTTER, J., (1981), "Flat Roofs", included in "*The Art of Construction*", September 1981.

ZMEUREANU, R., FAZIO, P., (1989), "Role of Computers in the Design of Energy-Responsive Buildings", *Canadian Journal of Civil Engineering*, Vol. 16, 1989, pp.55-61.

R3. CASE-BASED REASONING AND RELATED TOPICS

AAMODT, A., PLAZA, E., (1994), "Case-Based Reasoning: Foundational Issues, Methodological Variations and System Approaches", *AI Communications* 7(1), pp. 39-59, 1994.

AAMODT, A., (1994), "Explanation-driven case-based reasoning", in *Topics in Case-Based Reasoning* (editors: S.Weiss, K. Althoff, M. Richter), Springer Verlag, 1994, pp. 274-288.

AAMODT, A., (1991), "A Knowledge-intensive Approach to Problem Solving and Sustained Learning", *Ph.D. dissertation*, University of Trondheim, Norwegian Institute of Technology, May 1991. (University Microfilms PUB 92-08460)

AHA, D.W., (1997), "The Omnipresence of Case-Based Reasoning in Science and Application", *Technical report AIC-98-002*, Naval Research Laboratory, Navy Center for Applied Research in Artificial Intelligence, Washington, DC, 1997.

ALTHOFF, K. D., AURIOL, E., BARLETTA, R., MANAGO, M., (1995), "A Review of Industrial Case-Based Reasoning Tools, *AI Intelligence*, U.K., 1995

ASHLEY, K. D., (1989), "Assessing Similarities Among Cases – A Position Paper", *Proceedings of the DARPA Workshop on Case-Based Reasoning*, Pensacola Beach, FL, 1989, pp. 72-76.

ASHLEY, K. D., (1991), *Modeling Legal Arguments: Reasoning with Cases and Hypotheticals*. MIT Press, Bradford Books, Cambridge, 1991.

BAREISS, R., (1989), Exemplar Based Knowledge Acquisition: A Unified Approach to Concept Representation, Classification and Learning, Academic Press Inc., 1989.

BAREISS, R., KING, J. A., (1989), "Similarity Assessment in Case-Based Reasoning", *Proceedings of the DARPA Workshop on Case-Based Reasoning*, Pensacola Beach, FL, 1989, pp. 67-71.

BAREISS, R., PORTER, B., HOLTE, R., (1990), "Concept Learning and Heuristic Classification in Weak-Theory Domains", *Artificial Intelligence Journal*, volume 45 (no. 1-2), 1990, pp.229-264.

BARLETTA, R., (1991), "An Introduction to Case-Based Reasoning", *AI Expert*, August 1991.

CAUDILL, M., BUTLER, C., (1992), *Understanding Neural Networks: Computer Explorations*, The MIT Press, Cambridge, MA, 1992.

DOMESHEK, E., KOLODNER, J.L., (1992), "A Case-Based Design Aid for Architecture", *Artificial Intelligence for Design*, Editor: J. Gero, Kluwer, Boston, 1992.

FLEMMING, U., (1994), "Case-Based Design in the SEED System", *Knowledge-Based Computer-Aided Architectural Design*, G. Carrara and Y.E. Kalay (editors), Elsevier Science, pp. 69 – 91, 1994.

FREEMAN, J.A., SCAPURA, D.M., (1992), *Neural Networks: Algorithms, Applications, and Programming Techniques*, Addison-Wesley, Reading MA, 1992.

GEBHARDT, F., VOSS, A., GRATHER, W., SCHMIDT-BELTZ, B., (1997), *Reasoning with complex cases*, Kluwer Academic Publishers, March 1997.

GOEL, A., KOLODNER, J., PEARCE, M., BILLINGTON, M., ZIMRING, C., (1991), "ARCHIE: A Case-Based Architectural Design System", Technical Report GIT-CC-91/18, Georgia Institute of Technology, College of Computing, Artificial Intelligence Group, 1991.

GOEL, A., (1991), "A Model-Based Approach to Case Adaptation", *Proceedings of the Thirteenth Annual Conference of The Cognitive Science Society*, Lawrence Erlbaum Associates Inc., Publishers, Northvale, NJ, 1991.

GOEL, A., (1989), "Integration of Case-Based Reasoning and Model-Based Reasoning for Adaptive Design Problem Solving", *Ph.D. Thesis*, Department of Computer Science, The Ohio State University, 1989.

HAMMOND, K., (1989), *Case-Based Planning*, Perspectives in Artificial Intelligence, Academic Press, 1989.

HARMON, P., (1992), "Overview: Commercial Case-Based Reasoning Products", *Intelligent Software Strategies*, January 1992.

HENNESSY, D.H., HINKLE, D., (1992), "Applying Case-Based Reasoning to Autoclave Loading", *IEEE Expert Magazine* 7(5), pp: 21-26, October 1992.

HINRICHS, T. R., (1992*), Problem Solving in Open Worlds: A Case Study in Design*, Lawrence Erlbaum Associates Inc., Publishers, Northvale, NJ, 1992.

KOLODNER, J.L., (1993), *Case-Based Reasoning*, Morgan Kaufmann Publishers Inc., 1993.

KOLODNER, J.L., SIMPSON, R.L., (1989), "The MEDIATOR: Analysis of an early case-based problem solver", *Cognitive Science* 13(4), pp 507-549, 1989.

KOLODNER, J.L., (1989), "Judging Which is the "Best" Case for a Case-Based Reasoner", *Proceedings of the DARPA Workshop on Case-Based Reasoning*, Pensacola Beach, FL, 1989.

KOLODNER, J.L., (1983), "Maintaining Organization in a Dynamic Long-term Memory", *Cognitive Science*, vol.7, pp 243-280, 1983.

KOLODNER, J.L., (1983), "Reconstructive Memory, A Computer Model", *Cognitive Science*, vol.7, pp 281-328, 1983.

KOTON, P., (1989), "Using Experience in Learning and Problem Solving", *Ph.D. Thesis (October 1988)*, Laboratory of Computer Science, Massachusetts Institute of Technology, MIT/LCS/TR-441, 1989.

LEAKE, D.B., (1996), *Case-Based Reasoning: Experiences, Lessons and Future Directions, 1996.*

LENZ, M. (editor), (1998), "Case-Based Technology: From Foundations to Applications", Lecture Notes in Computer Science, 1998.

LOPEZ, B., PLAZA, E., (1993), "Case-Based Panning for Medical Diagnosis", *Methodologies for Intelligent Systems: 7th International Symposium, ISMIS '93*, J. Komorowski, Z. W. Ras (Editors.), pp. 96-105. Lecture Notes in Artificial Intelligence 689, Springer Verlag, 1993.

MAHER, M.L., de SILVA GARZA, G., (1997), "Case-Based Reasoning in Design", *IEEE Expert*, volume 12, no.2, March/April 1997.

MAHER, M.L., PU, P. (editors), (1997), Issues and Applications of Case-Based Reasoning in Design, 1997.

MAHER, M.L., BALANCHANDRAN, B., ZHANG, D.M., (1995*), Case-Based Reasoning in Design*, Lawrence Erlbaum Associates, Publishers, New Jersey, 1995.

MAHER, M.L., BALANCHANDRAN, B., (1994), "A Multimedia Approach to Case-Based Structural Design", *Journal of Computing in Civil Engineering*, 8(3), pp. 359-376, 1994.

MAHER, M.L., ZHANG, D.M., (1993), "CADSYN: A Case-Based Design Process Model", *Artificial Intelligence in Engineering, Design, and Manufacturing*, 7(2), pp. 97-110, 1993.

MAHER, M.L., ZHANG, D.M., (1991), "CADSYN: Using Case and Decomposition Knowledge for Design Synthesis", *Artificial Intelligence in Design*, J.S. Gero (editor), 1991.

MANAGO, M., ALTHOFF, K-D., TRAPHONER, R., (1993), "Induction and Reasoning From Cases", *European Conference on Machine Learning, Workshop on Intelligent Learning Architectures*, Vienna, April 1993.

OWEN, S., (1990), *Analogy for Automated Reasoning*, Perspectives in Artificial Intelligence, Volume 9, Academic Press, London, U.K., 1990.

PARSAYE, K., CHIGNELL, M., (1988), *Expert Systems for Experts*, John Wiley and Sons, 1988.

PORTER, B.W., BAREISS, R., HOLTE, R.C., (1990), "Concept Learning and Heuristic Classification in Weak-Theory Domains", *Artificial Intelligence*, volume 45, no. 1 & 2, 1990.

PURVIS, L., PU, P., (1998), "COMPOSER: A Case-Based Reasoning System for Engineering Design", *Robotica*, vol. 16, No. 3, pp. 285-295, May 1998.

RAO, V.B., RAO, H.V., (1993), *C++ Neural Networks and Fuzzy Logic*, MIS Press, 1993.

REICH, Y., (1993), "The Development of Bridger: A Methodological Study of Research on Machine Learning in Design", *Artificial Intelligence in Engineering*, 8(3), pp. 217-231, 1993.

REICH, Y., FENVES, S. J. (1995), "A System that Learns to Design Cable-Stayed Bridges", *Journal of Structural Engineering, ASCE*, 121(7), pp.1090-1100, July 1995.

RICHTER, A.M., WEISS, S., (1991), "Similarity, Uncertainty and Case-Based Reasoning in PATDEX". *Automated reasoning, essays in honour of Woody Bledsoe*, R.S. Boyer (editor), Kluwer, pp. 249-265, 1991.

RIESBECK, C.K., SCHANK, R.C., (1989), *Inside Case-Based Reasoning*, Lawrence Erlbaum Associates Inc., Publishers, Hillsdale, NJ, 1989.

RUMELHART, D.E., McCLELLAND, J.L., HINTON, G., (1986), "Learning Internal Representations by Error Propagation", *Parallel Distributed Processing: Exploration in the Microstructures of Cognition*, The MIT Press, Cambridge, MA, 1986.

SCHANK, R.C., (1982), *Dynamic Memory: A Theory of Learning in Computers and People*, Cambridge University Press, New York, 1982.

SHINGHAL, R., (1992), *Formal Concepts in Artificial Intelligence*, Chapman and Hall, 1992.

SIMPSON, R.L., (1985), "A Computer Model of Case-Based Reasoning in Problem Solving: An Investigation in the Domain of Dispute Mediation", *Technical Report GIT-ICS-85/18 (Ph.D. Thesis)*, Georgia Institute of Technology, School of Information and Computer Science, 1985.

SMITH, I., STALKER, R., LOTTAZ, C., (1996), "Creating Design Objects from Cases for Interactive Spatial Composition", *Artificial Intelligence in Design '96*, John S. Gero, Fay Sudweeks (editors), Kluwer Academic Publishers, pp. 97-116, 1996.

SKALAK, C.B., RISSLAND, E., (1992). "Arguments and Cases: An Inevitable Twining" *International Journal of Artificial Intelligence and Law,* 1(1), pp.3-48, 1992.

SYCARA, K., GUTTAL, R. KONING, J., NARASHIMHAN, S., NAVINCHANDRA, D., (1992), "CADET: a Case-Based Synthesis Tool for Engineering Design", *International Journal of Expert Systems*, volume 4, no. 2, 1992.

SYCARA, K.P., NAVINCHANDRA, D., (1989), "Integrating Case-Based Reasoning and Qualitative Reasoning in Design", *AI in Design*, Editor: J.Gero, Computational Mechanics, U.K., July 1989.

SYCARA, K.P., (1988), "Using case-based reasoning for plan adaptation and repair", *Proceedings of the DARPA Workshop on Case-Based Reasoning*, Clearwater Beach, FL, Morgan Kaufmann, pp. 425-434, 1988.

VELOSO, M., M., (1994), *Planning and Learning by Analogical Reasoning*, Lecture Notes in Artificial Intelligence series, Springer Verlag, Berlin, 1994.

WATSON, I. D., (1997), *Applying Case-Based Reasoning: Techniques for Enterprise Systems, 1997.*

APPENDIX A: SYSTEM PERFORMANCE CRITERIA

CRED employs the System Performance Criteria of ASHRAE 90.1/1989 as one of the major tests for checking if a building envelope is passing the design requirements.

To assess the energy efficiency of a new building through the System Performance Criteria, one must first define a theoretical model of the building, called the *prototype building,* whose parameters are based on general building characteristics (such as type, geometry, and orientation) and on average climatic data for the selected location. Then, to achieve the Standard-based requirements, the thermal performance of the new building design should be at least as good as the thermal performance of the associated prototype building:

$$H_{BLDG} + C_{BLDG} \leq H_{PROTOTYPE} + C_{PROTOTYPE}$$

where H_{BLDG}, C_{BLDG}, $H_{PROTOTYPE}$, and $C_{PROTOTYPE}$ are respectively the real wall heating load, the real wall cooling load, the prototype wall heating load (also called *heating criteria*) and the prototype wall cooling load (also called *cooling criteria*). Heating and cooling loads (either for real building on the prototype building) are calculated, respectively, with the heating and cooling equations presented below.

A1. HEATING EQUATION

$$H = \sum_{i=1}^{n} [\, HLUO_i + HLXUO_i + HLG_i + HLS_i + HLC_i + \sum_{j=1}^{m_i} (HLU_{ij} + HLM_{ij})]$$

where:

n is the number of wall orientations;

m_i is the number of wall mass types for ith orientation;

HLU, HLUO, HLXUO relate annual heating load with thermal transmittance of the wall;

HLM relates annual heating load with thermal capacity of wall;

HLG relates annual heating load with internal gains;

HLS relates annual heating load with incident solar gains;

HLC relates annual heating load with local weather data.

A2. COOLING EQUATION

$$C = \sum_{i=1}^{n} [CLUO_i + CLXUO_i + CLG_i + CLS_i + CLC_i + \sum_{j=1}^{m_i} (CLU_{ij} + CLM_{ij})]$$

where:

n is the number of wall orientations;

m_i is the number of wall mass types for ith orientation;

CLU, CLUO, CLXUO relate annual cooling load with thermal transmittance of the wall;

CLM relates annual cooling load with thermal capacity of wall;

CLG relates annual cooling load with internal gains;

CLS relates annual cooling load with incident solar gains;

CLC relates annual cooling load with local weather data.

The following weather parameters are used by the Standard to evaluate the climatic characteristics of the location:

HDD18: Heating degree-days below 18.3 C (65 F)

HDD10: Heating degree-days below 10 C (50 F)

CDD18: Cooling degree-days above 18.3 C

CDD10: Cooling degree-days above 10 C

CDH27: Cooling Degree-Hours above 26.67 C (80 F)

VSN: Annual average daily solar energy on North facade

VSEW: Annual average daily solar energy on East & West facades

VSS: Annual average daily solar energy on South facade

DR: Daily range average for temperatures of warmest month

For example, the values of these weather-related parameters, for Montréal, are as follows:

HDD18 = *4538*

HDD10 = *2439*

CDD18 = *178*

CDD10 = *1124*

CDH27 = *326*

VSN = *33.86*

VSEW = *73.51*

VSS = 104.12

DR = *9.87*

It should be noted that the requirements of the System Performance Criteria are achieved if, and only if, the sum of the heating and cooling loads are less than the sum of heating and cooling criteria, even if individual loads, either for heating or for cooling, are greater than in the corresponding prototype building.

REFERENCES

ASHRAE, (1989), "Energy Efficient Design of New Buildings Except New Low-Rise Residential Building", *Standard 90.1/1989*, American Society of Heating, Refrigerating and Air-Conditioning Engineers, 1989.

APPENDIX B: PROTOTYPE BUILDING ENVELOPE

The parameters recommended by the System Performance Criteria of ASHRAE 90.1/1989 for the theoretical model of the building (the so-called *prototype building*) are used by CRED not only to check the validity of the new design but also to infer missing information. Those parameters are as follows:

B1. MAXIMAL U-VALUE FOR THE ROOF

$$U_{ROOF} \leq \frac{5.68}{5.3 + 0.00324 * HDD18 + 0.00234 * CDD18 + 0.000468 * CDH27} \frac{W}{m^2 °C}$$

For Montréal:

$$U_{ROOF} \leq 0.276 \frac{W}{m^2 °C}$$

From Québec Law:

$$U_{ROOF} \leq 0.23 \frac{W}{m^2 °C}$$

B2. MAXIMAL U-VALUE FOR THE WALL BELOW GRADE

$$U_{WALL\ BG} \leq \frac{5.68}{4.5 + 0.0004167 HDD18} \frac{W}{m^2 °C}$$

For Montréal:

$$U_{WALL\ BG} \leq 0.89 \frac{W}{m^2 °C}$$

B3. MAXIMAL U-VALUE FOR THE WALL ABOVE GRADE

$$U_{WALL} \leq 159.92 \, x \, (HDD18)^{-0.712} \, \frac{W}{m^2 \, ^\circ C}$$

For Montréal:

$$U_{WALL} \leq 0.4 \, \frac{W}{m^2 \, ^\circ C}$$

B4. OPTIMAL PARAMETERS FOR THE GLAZING SYSTEM

$$U_{GLAZING} \leq 2.954 \, \frac{W}{m^2 \, ^\circ C}$$

$$\frac{GLAZING \; AREA}{WALL \; AREA} \leq 0.48 - 33.345 x \, 10^{-8} \, x \, CDD10 \, x \, VSEW \cong 0.275$$

From Québec Law:

$$U_{GLAZING} \leq 2.86 \, \frac{W}{m^2 \, ^\circ C}$$

$$\frac{GLAZING \; AREA}{WALL \; AREA} \leq 0.22$$

All climate related parameters (such as *HDD18*, *HDD10*, *CDD18*, *CDD10*, *CDH27*, *VSEW*) were explained in Appendix A which includes a detailed description of the System Performance Criteria from ASHRAE 90.1/1989.

REFERENCES

ASHRAE, (1989), "Energy Efficient Design of New Buildings Except New Low-Rise Residential Building", *Standard 90.1/1989*, American Society of Heating, Refrigerating and Air-Conditioning Engineers, 1989.

APPENDIX C: EXAMPLE OF BUILDING-PERFORMANCE SIMULATION

The tool selected to simulate the building-envelope performance is BESA, a program that could yield the complete thermal-performance analysis of a given prototype building that is specified along some general parameters such as building location, building geometry, total floor area, number of stories, building functionality (or type), and so forth.

The simulation sample presented below is for an office building. Additional data and missing information is inferred from either the ASHRAE Standard 90.1/1989 or from the Quebec Construction Law.

C1. BUILDING DESCRIPTION

C1.1. Primary Building Data

Location:	*Montréal*
Type:	*Office*
Number of Stories:	*10*
Floor Area:	*2560 m^2 per floor*
Orientation:	*Larger walls face East & West*

C1.2. Geometry Data

Floor Height:	*4 m*
Building Shape:	*rectangular with 1/2.5 aspect ratio*
Floor Dimensions:	*32m x 80m*
Wall Area:	*896 m^2/floor*
Wall-to-Floor Ratio:	*0.35*

C1.3. Internal Loads

Occupancy:	*25.55 m^2 per person*
Schedules of operation:	*from ASHRAE 90.1/1989*
Lighting:	*22 W/m^2 (Québec Law)*
Ballast Factor:	*1.2 (110 W lamps in BESA)*
Equipment:	*8 W/m^2*
Envelope	
Ground reflectivity:	*0.2 (from ASHRAE)*
Solar absorptivity:	*0.7 (from ASHRAE)*
Shading coefficient:	*0.7 (from ASHRAE)*
Glazing U-value:	*2.86 W/m^2C (Québec Law: R=0.35)*
Roof U-value:	*0.23 W/m^2C (Québec Law: R=4.4)*
Glazing-to-Wall ratio:	*22% (Québec Law)*
U-value for walls:	*0.4 W/m^2C (from ASHRAE)*

C1.4. Thermal Parameters

Heat Generation per Person:

Sensible heat gain:	*68 W/person (ASHRAE: 230 Btu/h)*
Latent heat gain:	*56 W/person (ASHRAE: 190 Btu/h)*

Thermostat Settings:

Cooling:	*24 degrees C (75 degrees F)*
Heating:	*21 degrees C (70 degrees F)*

Night Setback: *13 degrees C (55 degrees F)*

Ventilation Rate: *0.4 L/m2sec (ASHRAE: 10 L/s/person)*

Infiltration Rate: *0.17 ach (ASHRAE: 0.038 cfm/ft^2)*

C1.5. HVAC System

Packaged Rooftop VAV with Perimeter Reheat

C2. RUN ANALYSIS

Average Energy Consumption: *684 MJ/m^2 (190 kWh/m^2)*

Average Energy Cost: *11.81 $/m^2*

The dependency of the energy consumption of the building on the overall thermal resistance of the wall is shown on Table C-1.

Table C-1: Unit energy consumption function of wall R-value

WALL THERMAL RESISTANCE (m^{2}C/W)	ENERGY CONSUMPTION (kWh/m^2)
1.25	206.11
2.50	190.00
2.78	188.61
3.00	187.22
4.00	184.72
5.00	183.06
5.50	182.50
6.00	182.22
16.67	178.61

Statistical regression analysis (with STATGRAPHICS software package) of these data yields to the following fitted function:

$$Energy\ Consumption = 180.56 \times (1 + 0.345 \times e^{-0.723 \times R}) \quad \frac{kWh}{m^2}$$

where R is the thermal resistance of the wall (Figure C-1).

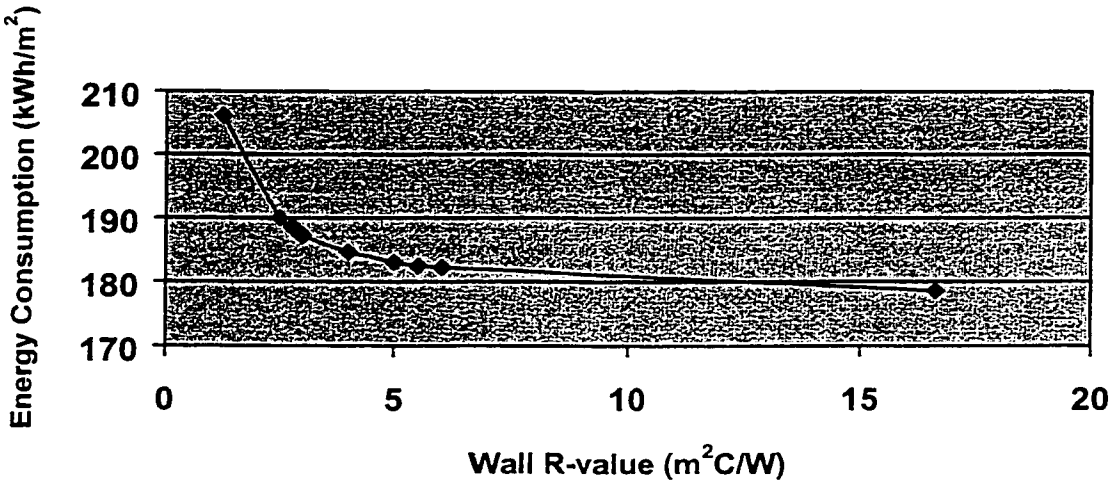


Figure C-1. Energy Consumption vs. Wall R-value

If the minimum R-value is considered to be 1.25 m^2C/W (50% less than ASHRAE recommendations of 2.5 m^2C/W), then the one obvious conclusion deriving from the above analysis is that the overall impact of the thermal characteristics of the wall on the energy performance of the building is limited to at most 13%.

Therefore, even if energy efficiency of the building is a desirable trait, increasing it through improvements in the thermal performance of exterior walls should be carefully considered. Analysis of energy savings should be accompanied by an analysis of material costs implied by any improvements.

REFERENCES

ASHRAE, (1989), "Energy Efficient Design of New Buildings Except New Low-Rise Residential Building", *Standard 90.1/1989*, American Society of Heating, Refrigerating and Air-Conditioning Engineers, 1989.

BESA 2.2, (1987), *Building Energy Systems Analysis*, CanadaPlan Group Inc, 1987.

APPENDIX D: LIFE CYCLE COST ANALYSIS

While attempting to repair a building envelope case failing the energy efficiency requirements, CRED limits the increase of the thermal resistance of the wall to at most 36% over the value recommended by the System Performance Criteria of ASHRAE 90.1/1989. The reasoning behind this limit is supported by the life cycle cost analysis that follows.

Considering the example of Appendix C, the maximum of energy savings that could be achieved through increasing the thermal resistance of the wall is:

$$MAXIMUM\ ENERGY\ SAVED = 180.56 \times 0.345 \times e^{-0.723R} < 62.3 \frac{kWh}{m^2\ year}$$

Studies have shown that the "present worth" of energy savings are given by a formula that takes into account the long-term value of money currently invested in the building envelope. It is possible to calculate the actual benefits of investing the money in an interest-bearing bank account instead and watching the investment grow over time. Therefore, the concept of *present worth (PW)* or *present value* should be introduced as:

$$PW = C\frac{1-(1+a)^{-n}}{a} \quad where \quad a = \frac{i-e}{1+e}$$

where n is the number of years under consideration, i is the cost of money (i.e. the average interest of banks), e is the rate at which the energy cost are expected to increase, and C is actual cost of energy saved per year.

The below formula defines a "present worth factor":

$$F = \frac{1-(1+a)^{-n}}{a}$$

Considering a lifetime estimation of 20 years (n = 20) and supposing that i = 8%, e = 5.5% we obtain a = 0.023, and, subsequently, F = 15.87.

The average cost of the energy per kWh, *EC*, may be approximated with (see in Appendix C the results yielded by BESA):

$$EC = 11.81/190 = 0.062 \ \$/kWh$$

Studying the price of different insulation materials, one notices that the cost of the insulation, *IC*, increases linearly with its thermal resistance, independent of the insulation type, according to the following relationship:

$$IC = 10 \times R \frac{\$}{m^2 \ of \ wall \ area}$$

where *R* is the insulation R-value in m^2C/W.

A case study carried out by the Masonry Council of Canada for which the average ratio of wall-to-floor area was assumed to be 0.36, concluded that the insulation cost (*IC*) can be roughly approximated by:

$$IC = 3.6 \times R \frac{\$}{m^2 \ of \ floor \ area}$$

Hence, the real savings in cost through the increase of the thermal resistance of the wall are given by the long-term savings of energy costs (over the estimated period of 20 years) minus the losses due to additional insulation costs:

$$TOTAL \ COST \ SAVED = 61.3 \times e^{-0.723R} - 3.6 \times R \quad [\$]$$

where

$$61.3 = 180.56 \times 0.345 \times EC \times F$$

The optimal thermal resistance of the wall, at which maximum cost savings are to be obtained over a span of 20 years, is given by:

$$R_{opt} = \frac{1}{0.723} \ln \frac{61.3}{3.6} = 3.92 \ \frac{m^2 °C}{W}$$

Since the r-value recommended by the ASHRAE Standard 90.1/1989 is

$$R_{ASHRAE} = \frac{1}{0.4} = 2.5 \ \frac{m^2 °C}{W}$$

then the maximum variation range for the wall R-value is [2.5, 3.92] or 36%.

REFERENCES

Masonry Council of Canada, Guide to Energy Efficiency in Masonry and Concrete Buildings, 1982.

APPENDIX E: MATHEMATICAL EQUATIONS USED TO SIMULATE THE BUILDING ENVELOPE BEHAVIOR

To simulation the performance of a building envelope case, CRED needs to compute the transversal heat and vapor flow across the envelope components. For that purpose, CRED employs the set of mathematical equations shown in this appendix.

E1. TEMPERATURE GRADIENT ACROSS THE BUILDING ENVELOPE

The overall heat flow traversing the building envelope, in steady-state conditions, is directly proportional to the magnitude of the total temperature difference and the thermal transmittance (i.e. inversely proportional to the thermal resistance):

$$Heat\ Flow \propto \frac{Temperature\ Difference}{Thermal\ Resistance}$$

It follows that temperature drop through each component of the envelope is proportional to the component's thermal resistance. The temperature gradient through the *i*-th layer may be computed, in steady-state conditions, with the following formula:

$$dt_i = R_i \times \frac{\Delta T}{\sum_i R_i}$$

where:

dt_i = temperature drop across layer *i*,

R_i = thermal resistance of layer *i*.

E2. WATER VAPOR DIFFUSION ACROSS THE BUILDING ENVELOPE

The water vapor pressure difference that exists between material surfaces and the permeability of the material determines the water vapor diffusion across a building material. In general, the permeability of a building material subjected to a pressure difference varies with the relative humidity of the material, but for most practical situations, the material permeability could be considered constant.

The water vapor flow produced through diffusion can be determined, assuming steady-state flow conditions, with the following formula:

$$W = \frac{\mu(\Delta p)}{L} = \frac{\Delta p}{R}$$

The water vapor pressure drop through each component of the envelope is proportional to the component's water vapor resistance. The water vapor gradient through the *i*-th layer may be computed, in steady state conditions, with the following formula:

$$dp_i = R_i \, x \, \frac{\Delta p}{\sum_i R_i}$$

where:

dp_i = water vapor pressure drop across layer *i*,

R_i = water vapor resistance of layer *i*.

E3. PROPERTIES OF ATMOSPHERIC AIR

Temperature:

$$T = temperature + 273.15 \ ^\circ K$$

Density:

$$\rho = \frac{p}{RT} = \frac{101325}{287.1}\frac{1}{T} = \frac{352.93}{T}\ \frac{kg}{m^3}$$

Coefficient of thermal expansion:

$$\beta = -\frac{1}{\rho}\frac{\delta\rho}{\delta T} = \frac{1}{T}\ \frac{1}{°K}$$

Conductivity:

$$k = \frac{2.6443\sqrt{T}}{1 + \frac{245}{T}10^{\frac{12}{T}}}10^{-3}\ \left[\frac{W}{m°K}\right]$$

The approximation is valid when air temperature is within the range of -183 ... 900 degrees Celsius.

Dynamic air viscosity:

$$\mu = (c_0 + c_1 T + c_2 T^2 + c_3 T^3 + c_4 T^4)10^{-7}\ \frac{kg}{m\,\mathrm{sec}}$$

where

$c_0 = 4.0201$,

$c_1 = 7.4582 \times 10^{-1}$,

$c_2 = -5.7171 \times 10^{-4}$,

$c_3 = 2.9928 \times 10^{-7}$,

$c_4 = -6.2524 \times 10^{-11}$.

The approximation is valid when air temperature is within the range of -53 ... 1577 degree Celsius. The above equation was produced through statistical analysis and it fits the experimental data with a precision of 2%.

Kinematic air viscosity:

$$\nu = \frac{\mu}{\rho} = \frac{\mu T}{352.93} \left[\frac{m^2}{s}\right]$$

Grashof number:

$$Gr = \frac{g\beta \rho^2 (\Delta T) L^3}{\mu^2}$$

where L is the width of air space, or the height of air film, in mm.

Prandtl number:

$$Pr = \frac{c_p \mu}{k}$$

where c_p is 1005 J/kgC.

Rayleigh number:

$$Ra = Pr \, x \, Gr = \frac{g\beta \rho^2 (\Delta T) L^3 c_p}{\mu k}$$

Nusselt number:

$$Nu = \frac{h_c L}{k}$$

where h_c is the coefficient of convective heat transfer.

References

ROHSENOW, W. M., HARTNETT, J. P., (1973), *Handbook of Heat Transfer*, chapter 2: Thermophysical Properties, page 2-80, McGraw-Hill, 1973.

VINES, R. G., (1960), "Measurement of Thermal Conductivity of Gases at High Temperature", *Journal of Heat Transfer*, Transactions ASME, 82:48, 1960.

E4. PRESSURE OF WATER VAPORS AT SATURATION

Fitting experimental data for temperature of water vapors within the range of -100 ... 200 degrees Celsius is calculated by using the following equation:

$$\ln(p_{sat}) = \frac{c_1}{T} + c_2 + c_3 T + c_4 T^2 + c_5 T^3 + c_6 T^4 + c_7 \ln(T) \quad Pa$$

where the coefficients c are given in the table shown below.

Table E-1: Coefficients used for vapor pressure at saturation

	$-100\,°C < temperature < 0\,°C$	$0\,°C <= temperature < 200\,°C$
c_1	-5674.5359	-5800.2206
c_2	6.3925247	1.3914993
c_3	$-0.9677843 \times 10^{-2}$	-0.04860239
c_4	$0.62215701 \times 10^{-6}$	$0.41764768 \times 10^{-4}$
c_5	$0.20747825 \times 10^{-8}$	$-0.14452093 \times 10^{-7}$
c_6	$0.9484024 \times 10^{-12}$	0
c_7	4.1635019	6.5459673

References

WEXLER, A., HYLAND, R. W., (1980), "A Formulation for Thermodynamic Properties of the Saturated Pure Ordinary Water Substance from 173.15 to 473.15 degrees Kelvin", Final Report, *ASHRAE Research Project RP216*, 1980.

E5. HEAT TRANSFER COEFFICIENT FOR OUTSIDE AIR FILM

The outside air-film coefficient, accounting for convective and radiative heat transfers occurring at the outer surface of the envelope assembly, is calculated using the following formulas taken from the ASHRAE Handbook of Fundamentals (1989):

E5.1. Winter Coefficient

$$f_{out} = 34.08 \ \frac{W}{m^2 °C} \quad \text{(Wind speed = 6.7 m/sec)}$$

E5.2. Summer Coefficient

$$f_{out} = 22.72 \ \frac{W}{m^2 °C} \quad \text{(Wind speed = 3.4 m/sec)}$$

E6. HEAT TRANSFER COEFFICIENT FOR INSIDE AIR FILM

The inside air-film coefficient should account for both convective and radiative heat transfers occurring between the inner surface of the envelope assembly and the room air. Hence, the thermal conductance of the so-called *still air* is calculated using the following formula taken from the ASHRAE Handbook of Fundamentals (1989):

$$f_{in} = h_c + h_r = 1.77 (T_R - T_S)^{0.25} + \sigma \varepsilon \frac{T_R^4 - T_S^4}{T_R - T_S} \ \frac{W}{m^2 °C}$$

where

σ is the Boltzmann constant

T_R, T_S are respectively the room and surface temperatures,

ε is the emissivity of the inner surface.

Since the surface temperature is necessary to calculate the conductance of still air, the computation of f_{in} is an iterative process that will start with an initial approximation of f_{in} and will continue with successive determinations of T_S and f_{in}, till the differences between two consecutive iterations will fall below a given threshold.

Better approximations for the coefficient of convective heat transfer, h_c, are based on the Nusselt number and take into account the height of the air film and its inclination with respect to the vertical:

$$h_c = \frac{Nu\ x\ k}{H}$$

E6.1. Vertical Plate in Laminar Range

$$Nu = 0.68 + \frac{0.67\ Ra^{\frac{1}{4}}}{\left[1 + \left(\frac{0.492}{Pr}\right)^{\frac{9}{16}}\right]^{\frac{4}{9}}} \quad (0 < \mathrm{Ra} < 10^9):$$

E6.2. Vertical Plate Outside the Laminar Range

$$Nu = \left[0.825 + \frac{0.387\, Ra^{\frac{1}{6}}}{\left(1 + \left(\frac{0.492}{Pr} \right)^{\frac{9}{16}} \right)^{\frac{8}{27}}} \right]^2$$

E6.3. Slanted Plate with Warmer Inner Side (heat loss from the room)

$$Nu = 0.56\,(Ra\cos(\theta))^{\frac{1}{4}}$$

E6.4. Horizontal Plate with Warmer Inner Side (heat loss from the room)

$$Nu = 0.58\,(Ra)^{\frac{1}{5}}$$

The equations for plates with warmer outer side (i.e. when the room gains heat) are not presented since in those cases we expect to have no moisture-induced problems.

References

CHURCHILL, S. W., CHU, H. H. S., (1975), "Correlating Equations for Laminar and Turbulent Free Convection from a Vertical Plate", *International Journal of Heat Mass Transfer*, volume 18, p.1323, 1975.

FUJII, T., IMURA, H., (1972), "Natural Convection Heat Transfer from a Plate with Arbitrary Inclination", *International Journal of Heat Mass Transfer*, volume 15, p.755, 1972.

INCROPERA, F.P., DeWITT, D.P., (1981), *Fundamentals of Heat Transfer*, chapter 9: Free Convection, John Wiley, 1981.

E7. HEAT TRANSFER COEFFICIENT FOR AIR SPACE CAVITY

As in the case of inside air film, the air space coefficient is the sum of convective and radiative heat transfer coefficients:

$$h = h_c + h_r = \frac{k\,Nu}{L} + \sigma \frac{1}{\frac{1}{\varepsilon_1} + \frac{1}{\varepsilon_2} - 1} \frac{T_1^4 - T_2^4}{T_1 - T_2} \quad \frac{W}{m^2 °C}$$

where indices 1 and 2 represents the two boundary surfaces.

E7. 1. Vertical Air Space

For Ra < 2000:

$$Nu = 1.0$$

For 2000 <= Ra < 200000:

$$Nu = 0.197\,(Ra)^{\frac{1}{4}}\,(H / L)^{-\frac{1}{9}}$$

For Ra >= 200000:

$$Nu = 0.073\,(Ra)^{\frac{1}{3}}\,(H / L)^{-\frac{1}{9}}$$

E7.2. Horizontal Air Space

For Ra < 1700:

$$Nu = 1.0$$

For 1700 <= Ra < 7000:

$$Nu = 0.059(Ra)^{0.4}$$

For 7000 <= Ra < 320000:

$$Nu = 0.212(Ra)^{\frac{1}{4}}$$

For Ra >= 320000:

$$Nu = 0.061(Ra)^{\frac{1}{3}}$$

E7.3. Slanted Air Space

For inclination angles with respect to horizontal in the range of 0 ... 70 degrees:

$$Nu = 1 + 1.44\left[1 - \frac{1708}{Ra\cos(\theta)}\right] \times \left[1 - \frac{1708\sin(1.8\theta)^{1.6}}{Ra\cos(\theta)}\right] + \left[\left(\frac{Ra\cos(\theta)}{5830}\right)^{\frac{1}{3}} - 1\right]$$

For inclination angles in the range of 70 ... 90 degrees:

$$Nu = N_{u90°}(\sin(\theta))^{\frac{1}{4}}$$

References

ARNOLD, J. N., CATTON, I., EDWARDS, D.K., (1975), "Experimental Investigation of Natural Convection in Inclined Rectangular Regions of Differing Aspect Ratio", *ASME Paper* 75-HT-62, 1975.

AYYASWAMY, P.S., CATTON, I., (1973), "The Boundary Layer Regime for Natural Convection in a Differentially Heated Tilted Rectangular Cavity", *International Journal of Heat Transfer*, volume 95, p.543, 1973.

HOLLANDS, K.G.T., UNNY, S.E., RAITHBY, G.D., KONICEK, L., (1976), "Free Convective Heat Transfer Across Inclined Air Layers", *International Journal of Heat Transfer*, volume 98, p.189, 1976.

HOLMAN, J.P., (1981), *Heat Transfer*, Fifth Edition, chapter 7: Natural Convection Systems, McGraw-Hill, 1981.

INCROPERA, F.P., DeWITT, D.P., (1981), *Fundamentals of Heat Transfer*, chapter 9: Free Convection, John Wiley, 1981.

APPENDIX F: ARTIFICIAL NEURAL NETWORK FILTER

This research focused on back-propagation feed-forward layered neural networks with one or two hidden layers. The mathematical equations and the training heuristics used for the ANN implementation in CRED are listed in this appendix.

F1. FUNDAMENTALS

Activation function is logistic (sigmoid):

$$f(x) = \frac{1}{1 + e^{-\alpha x}}$$

Output function is usually the identify function:

$$f(x) = x$$

Differential of input-output function is:

$$d[f(x)] = cf(x)(1 - f(x))$$

The generalized *Delta Rule* for weight adjustment in back-propagation is:

$$\Delta w_{ik}^{L} = \beta E_k^L x_i^L + \alpha(\Delta w_{ik}^{L})_{PREV}$$

where β is the learning rate constant and α is the momentum constant.

F2. TRAINING ALGORITHM

Let I be an input specimen generating the output O instead of target value T when fed into a neural network. The training algorithm is shown below.

I. Initialize weights w_{ik} to random values.

II. Feed one training specimen to the neural network.

III. Calculate the output error for each cell k of the output layer L as follows:

$$E_k^L = d[f(I_k^L)](O_k^L - T_k^L)$$

IV. Begin back propagation. For each cell k of middle layer L calculate the error as follows:

$$E_k^L = d[f(I_k^L)]\sum_i w_{ik}^{L+1} E_i^{L+1}$$

V. Compute the amount of weight adjustment function of the previous adjustment value as follows:

$$\Delta w_{ik}^L = \beta E_k^L x_i^L + \alpha(\Delta w_{ik}^L)_{PREV}$$

VI. For case updating, update the cell weights as follows:

$$w_{ik}^L = w_{ik}^L + \Delta w_{ik}^L$$

VII. For epoch updating, accumulate the weight adjustments for each training specimen:

$$c_{ik}^L = c_{ik}^L + \Delta w_{ik}^L$$

VIII. At the end of one training pass, for epoch updating, adjust the cell weights as follows:

$$w_{ik}^L = w_{ik}^L + c_{ik}^L$$

IX. At the end of one training pass, if the overall error is below the preset threshold, then stop. Otherwise, restart feeding the network with training specimen.

F3. TRAINING HEURISTICS

The number of passes required to train the neural network and the quality of the training is determined by:

- the learning rate constant,
- the momentum constant,
- the size of the middle layer with respect to the input layer,

- the initial weight values
- the amount of noise added to the training specimen, and
- the size of the training set.

A set of heuristic relationships between the training parameters was identified by applying regression analysis to the training behavior of networks with various configurations. For a given network configuration, and the same initial values of cell weights the conclusions are:

- **Learning Rate**
  - In the absence of noise, the number of passes to train decreases following a non-linear function with the increase of the learning rate.
  - Learning rate should be small (0.1 ... 0.2) to avoid high oscillation of the back propagated error (due to large modifications applied to the cell weights), and to increase the chances of reaching the absolute minimum during the gradient-descent learning. Making the learning constant too small may exponentially increase the learning rate.
  - The learning rate should be decreased when the amount of noise is larger. It is unusual but quite possible to dynamically lower the learning constant as the training progresses.
- **Momentum Constant**
  - The number of passes necessary to train the network is linearly dependent on the value of the momentum constant (if that one is below 0.9). The momentum constant should be kept at a higher value (i.e. close to 0.9) to compensate for lower learning

constants and to speed up the training. Increasing the momentum above 0.9 does not improve the speed of training and may adversely affect the gradient descent learning.

- **Middle Layer Size**
  - Apparently, the number of passes to train increases following a non-linear relationship with the size of the middle layer. The time-per-training pass increases as well, due to the linear increase of the number of weights to adjust.
  - The size of the middle layer should be significantly smaller in size than the input layer to extract the features of the training specimen. An optimum size of the middle layer increases the noise rejection and keeps the training time within reasonable limits. Networks with middle layers that are too small either do not train or tend to over-generalize the training specimen, and, consequently, do not clearly separate the classes.
- **Error Threshold**
  - It is impossible to say if the absolute minimum was achieved as long as the error is below the set threshold. Hence, the set threshold should be carefully selected.
- **Training Noise**
  - The number of passes to train increases following a non-linear relationship with the amount of noise added to the training specimen.
  - Adding noise to the training specimen improves the robustness of the network by artificially enlarging the training set. The smaller the training set, the higher should be the amount of added noise (as high as 60 to 80%). Too much noise, however, may thwart the training.

- Minimizing the learning constant not only optimizes the gradient-descent learning but also allows more passes to better cover noisy specimen.
- It is recommended to gradually reduce the noise amount as the training progresses.

REFERENCES

CAUDILL, M., BUTLER, C., (1992), *Understanding Neural Networks: Computer Explorations*, The MIT Press, Cambridge, MA, 1992.

FREEMAN, J.A., SCAPURA, D.M., (1992), *Neural Networks: Algorithms, Applications, and Programming Techniques*, Addison-Wesley, Reading MA, 1992.

RAO, V.B., RAO, H.V., (1993), *C++ Neural Networks and Fuzzy Logic*, MIS Press, 1993.

RUMELHART, D.E., McCLELLAND, J.L., HINTON, G., (1986), "Learning Internal Representations by Error Propagation", *Parallel Distributed Processing: Exploration in the Microstructures of Cognition*, The MIT Press, Cambridge, MA, 1986.

APPENDIX G: AUTOMATED REASONING (RULE-BASED) MODULE

CRED uses a rule-based engine for adapting and repairing building envelope cases. The rule-based part of CRED, the so-called the *Automated Reasoning Module* (ARM), was designed to support two operating modes:

1) *Resolution Refutation*, where a set of input clauses is processed through resolution refutation, until either the desired goal is reached (the theorem is proved) or no more resolutions are possible, and
2) *Expertise Modeling*, where a set of production rules (prodrules) are sequentially fired, to infer new facts from the existing ones, until either the solution of the reasoning problem is achieved or no more rules may be fired.

Each mode of operation is further split, based on the constraints established at the very beginning, into *running sessions*. Hence, ARM may function in one of the following four running sessions:

1) *Resolution refutation toward a goal*,
2) *Resolution refutation toward answering questions*,
3) *Expertise modeling with "regular" forward or backward chaining*,
4) *Expertise modeling with plausible reasoning*.

The session type is defined at the beginning, when the problem to solve is loaded.

G1. FUNDAMENTALS

- *Rules of inference are used to deduce consequences from premises* (valid for both propositional and predicate logic). The list of such rules includes:

- Introducing conjunction,
- Eliminating conjunction,
- Introducing disjunction,
- Modus ponens,
- Modus tollens,
- Chaining,
- Universal specialization (or instantiation).

- *Formulae are converted into clauses.* A *clause* is a disjunction of zero or more literals, and a *literal* is an atom or a negation of an atom.
- *Clauses are standardized* (no two clauses contain the same variable).
- The resolution-refutation procedure includes the following steps:
  - The input set is assembled from the premises and the goal converted into clause form.
  - Two clauses selected to be resolved are first standardized.
  - The equality literal is handled through demodulation and paramodulation.

The resolution is refined with ordering, pruning, and restriction strategies.

G2. RESOLUTION STRATEGIES

The strategies implemented by CRED are shown in Table G-1. The different types of strategies are combined to obtain a more complex strategy such as "Breadth-First with Unit Resolution and Negative Support" or "Fewest Literals with Unit Resolution and Positive Support".

Table G-1. Resolution strategies implemented by CRED

Strategy Class	Strategy Name
ORDERING	Breadth-First (Depth Saturation)
	Fewest Literals Preference
	Weighting
PRUNING	Deleting Chaste Clauses
	Deleting Tautologies
	Deleting Subsumed Clauses
	Evaluating Ground Literals
RESTRICTION	Goal Supported Strategy
	Negative Resolution
	Positive Resolution
	Negative Hyper-resolution
	Positive Hyper-resolution
	Unit Resolution
	SLD (Linear Resolution with Selector on Definite Clauses)
	LBS (linear Back-Chaining with Selector on Assertions)

G3. BASIC DATA STRUCTURES

For resolution refutation, ARM employs three main data structures as follows:

- *TERM*, referring not only to constants, variables, and functions but also to predicates (any term is labeled as either *CONSTANT*, *VARIABLE*, *FUNCTION*, or *PREDICATE*),
- *CLAUSE*, which represents a clause as a set of literals (where a literal is a *TERM* of *PREDICATE* type), and
- *SUBSTITUTION*, which is a pair of terms, one being the term to be introduced, and the other one the term to be replaced.

Both *TERMs* and *SUBSTITUTIONs* are not stand-alone data structures but linked within the *CLAUSE* using them. Clauses are grouped into four lists identified as follows:

- *DEPTH* list: clauses are ranked according to their generation order (their ID numbers). Earlier generated clauses (lower ID numbers) are closer to the head of the list.
- *LENGTH* list: clauses are ranked based on their length, where the length of a clause is defined by the number of literals within the clause. Shorter clauses are placed ahead of longer clauses. Hence, the NIL clause will always be found at the head of that list.
- *SET_OF_SUPPORT* list: clauses that are part of the set-of-support. Defining which clauses are parts of the set-of-support depends on the strategy selected.
- *CONSISTENT* list: clauses that cannot be part of the SET_OF_SUPPORT list.

All clauses are to be found in both the *DEPTH* and *LENGTH* lists but only in one of the *SET_OF_SUPPORT* or *CONSISTENT* lists. The *SET_OF_SUPPORT* and *CONSISTENT* are complementary (if a clause is in one of the two lists, then it is not in the other list).

There are four types of *set-of-support* options defined as follows:

- *Goal Support*: the set-of-support is derived from the goal clause (which is actually the negation of the goal). In the *SET_OF_SUPPORT* list will be found, in addition to the goal clause, all the clauses having the goal clause among their ancestors. The remaining clauses are to be found in the *CONSISTENT* list. It is obvious that this kind of support may be employed only if the running session is one of resolution refutation toward a goal (no tautologies found in the input set of clauses).
- *Positive Support*: the set-of-support is provided by the positive clauses. Hence, if a clause is positive it is to be found in the *SET_OF_SUPPORT* list; otherwise, (a mixed or negative clause) it is to be found in the *CONSISTENT* list.

- *Negative Support*: the set-of-support is provided by the negative clauses. A negative clause is to be found in the *SET_OF_SUPPORT* list and a mixed or positive clause is to be found in the *CONSISTENT* list.
- Weight Support: the hierarchy of clauses is based on their weight values. This is not a set-of-support type in the real sense of its definition, but an extension of it to facilitate the access to the *CONSISTENT* list, when the weighting strategy is employed. When the weighting strategy is used, all clauses are pushed into the *CONSISTENT* list (the *SET_OF_SUPPORT* list will remain empty), and ranked so that the clauses with higher weight values are closer to the head of the *CONSISTENT* list.

G4. IMPLEMENTATION DETAILS

During the resolution refutation procedure, as well as in the expertise modeling, the processing of the clauses includes the following operations (or strategies):

- **Chaste clauses are deleted** (a chaste clause being defined as a clause containing a chaste literal, while a chaste literal is a literal for which no complementary unifiable literal may be found anywhere within the set of clauses). The operation to delete the chaste clauses is performed only once, at the very beginning (immediately following the loading of the input set of clauses). In forward chaining, where the goal is not known at the very beginning, the chaste clauses are not deleted.
- **Tautologies are deleted.** The only tautologies preserved are those present in the input set and having exactly two literals (one positive and the other negative). That type of tautology is to be considered a question to be answered.

- **All duplicate or subsumed clauses are deleted.**
- **Ground literals are evaluated and deleted (if false).** When a ground literal is found to be true, then the clause can be deleted as a tautology. This option is dismantled during SLD and LBS strategies, to avoid some unwanted side effects such as backtracking to a shallower than expected level.
- **Clauses are factorized** (and all subsumed literals are deleted).
- **Clauses are standardized,** so that no two clauses share the same variable.
- **Hyper-resolution is employed whenever it is possible** (whenever more than one satellite may be found).
- **All clauses containing functions** (shown by the nesting level of the clause stored in field *deep* of the CLAUSE data structure) **are forward demodulated or paramodulated,** if that is possible.
- If a newly generated clause (called *resolvent*) contains a non-negated equal predicate, then an attempt is made to use this clause either as a demodulator (if it is an unit clause) or as a paramodulator for backward demodulation or paramodulation.
- If a newly generated clause is a unit clause whose single literal is grounded, then an attempt is made to use this clause for reducing that ground literal from all previously generated clauses. If the newly produced ground literal is located in any of the previous clauses, then that clause is marked as *DEAD* (and eventually deleted), and it is replaced by an instance of it from which the ground literal was deleted (only if the ground literal found within the clause evaluates to FALSE).

Some remarks about the type of hyper-resolution employed are required. Most of the time, we have either a negative or positive hyper-resolution, where a nucleus is associated with a set of either negative or positive satellites. However, a third type of hyper-resolution is often allowed, called *mixed hyper-resolution*, where within the set of satellites involved in the resolution, there are both negative and positive unit clauses (a constraint was adopted for mixed hyper-resolution such that the satellites should be unit clauses).

ARM includes some built-in "knowledge" such as:

EQUAL[x,x] is always TRUE,

EQUAL[x,y] is identical to EQUAL[y,x]

Once a new clause is produced, all the required checking for the clause is performed, including factorization, elimination of evaluated ground literals, testing if the clause is not subsumed by a previous one, and so forth. If the clause passes the whole checking procedure, then it is declared to be valid and it is inserted in three of the four lists (*DEPTH*, *LENGTH* and one of either *SET_OF_SUPPORT* or *CONSISTENT* lists); otherwise the clause is deleted and its storage de-allocated.

ARM has an embedded "daemon" triggered whenever some meaningful events are happening such as passing to a new depth level in the breadth-first strategy or to a new length level in the fewest literal strategy. This daemon will start a global ground elimination and demodulation/paramodulation process meant to promote the resolution refutation by doing some additional simplifications to the existing clauses. For example, some of the previous demodulators or paramodulators may have been later simplified (through reduction of ground literals and additional demodulation or paramodulation), so that it is fully justified to re-use them for a new backward demodulation or paramodulation process.

The resolution refutation stops successfully when either NIL clause is generated (always found at the head of *LENGTH* list) or all the questions asked in the input set have been answered. The resolution refutation returns with a failure message when the problem was not solved and no more resolvents may be generated.

G5. EXPERTISE MODELING

In the expertise-modeling mode of operation, ARM acts like an *inference engine*, trying to prove consequents given a set of antecedents and a set of rules which define the truthfulness or falseness of each consequent in terms of the antecedents.

Rules (also called *prodrules*) are defined as a set of antecedents paired with a consequent. Each antecedent and consequent is a simple statement, consisting of leading *keyword* followed by a predicate (or literal). The most general form of a prodrule supported by this inference engine is the following:

IF	*literal(1)*	antecedent
AND	*literal(2)*	
.....		
AND	*literal(9)*	antecedent
THEN	*literal(0)*	consequent
WITH	*p*	certainty value of the rule ($0 < p < 1$)
STOP		optional (stopping the inference when fired)

Any of the statements starting with AND, WITH, or STOP keywords are optional. If the WITH statement is not in place, then the certainty value of the prodrule is assumed to be 1. If the STOP keyword is present in the prodrule layout, then the prodrule is called a *stop prodrule* because it will stop the inference engine after it fires (in forward chaining only). The prodrules of this type (with both antecedents and consequent) are called *regular prodrules*. There are some other, more

simplified forms, of prodrules, called facts. A *fact* is defined as a prodrule without any antecedent, and it is identified by the keyword IS. The general form of a fact is shown below:

IS *literal(0)*
WITH *p*

Any prodrule (or fact) is transformed into clause having the following form:

literal(0) + ~literal(1) + ... + ~literal(9)

That clause is associated with the prodrule through a pointer linkage (see field *clause* in the RULE data structure). Apparently, the clause linked to a prodrule is a definite clause, but this is not a requirement. The forward chaining may succeed even without a set of definite clauses. However, in backward chaining, which employs the LBS strategy, all the clauses associated with prodrules should be definite clauses. This set of definite clauses will be transformed into a set of Horn clauses, by appending the negation of the goal that is not associated with any prodrule. It is obvious, from the above definitions, that facts are linked to unit clauses while regular prodrules are linked to longer clauses (having at least two literals).

To prevent the stalling of the inference engine (when no prodrules can fire), default prodrules are sometimes necessary. A default prodrule is similar to a fact, in the sense that it has no antecedent (or, in other words, its antecedent is always true), but it will not be fired unless no other prodrule can fire. A default prodrule is introduced by the keyword ISD, as follows:

ISD *literal(0)*
WITH *p*
STOP

For problems to be solved in backward chaining, a goal is required. The goal is introduced by the keyword IS? as it is shown below:

IS? literal(0)

The goal is transformed into a negative unit clause (the negation of the goal), and it is stored not as a rule but only as a clause (to participate in LBS resolution but to have no part in any form of forward chaining).

When any of the certainty values p is less than 1, we have a plausible reasoning session requiring that all the clauses concluding on the same fact should be fired (to get the cumulative certainty value of the inferred fact).

Rules are linked into four different lists, in accordance with their *status* (see *status* field in the RULE data structure). A rule may be found in one, and only one, of the following four lists:

- *UNUSED* : some of the prodrule's antecedents are yet to be known
- *CHILLED* : prodrule was chilled (it is refractory or it has a false antecedent)
- *HEATED* : all the antecedents are true and prodrule is ready to fire
- *FIRED* : prodrule was fired

In each of these lists (with the exception of the *FIRED* list), the rules are organized according to the following principles:

- A fact is placed closer to the head of the list than a regular prodrule.
- A default prodrule is always pushed at the tail of the list.
- A prodrule having a given literal among its antecedents is always placed after (closer to the tail of the list) a prodrule having that literal in its consequent; in other words, the prodrules concluding on some given facts are placed ahead of the prodrules using those facts.

- Prodrules concluding on same facts are clustered together, if that is possible (to fire them one after the other in plausible reasoning).
- When all the above-mentioned guidelines are respected, a longer regular prodrule (supposed to be more specific) is placed ahead of a shorter regular prodrule (here we discuss regular prodrules, which do not include facts).
- Prodrules, having in their antecedent more recently inferred facts, are placed ahead of prodrules with less recently inferred facts in their antecedent; the recency of a fact may be assessed by exploring the position of the associated prodrule within the FIRED list.

The organization of the *FIRED* list is based on the firing order of the prodrules, therefore, the most recently inferred facts are linked to the prodrules placed at the tail of the *FIRED* list.

Whenever a prodrule is fired (meaning that all its antecedents are true), the new inferred fact, derived from its consequent, is transformed into a unit clause and checked as any other clause. If it duplicates one of the existing unit clauses, it is deleted but not before the weight of the existing unit clause is accordingly updated.

The computation of the certainty value of any new fact is performed when the prodrule is heated; that is, the prodrule is moved into the *HEATED* list, and the propagation of this certainty value (if more than one prodrule concluded on the same fact) is achieved when the prodrule is actually fired; that is, the prodrule is moved into the *FIRED* list.

The two strategies employed by the inference engine are backward chaining and forward chaining. Backward chaining has two alternatives: one in which the inference is stopped immediately after the first answer is reached, and the second in which all possible answers are delivered.

G5.1. Backward Chaining Algorithm

I. Append the negation of the goal (the question asked) to the set of clauses associated with prodrules.

II. Delete all chaste clauses (to simplify the inference).

III. Employ the LBS strategy to solve the problem (force LBS to return all possible answers if it is a plausible reasoning session).

IV. WHILE (LBS resolution refutation fails) DO

- Enable one of the default prodrules (the first one), by making its fact available to the resolution refutation procedure (default prodrule is transformed into a fact).
- If no default prodrule is found, then return with a failure message.

V. If no solution was found by the LBS strategy, then return failure; otherwise, mark all clauses leading to the proof and continue.

VI. For all the clauses leading to the proof, mark the associated prodrules (if any) by moving them into the *CHILLED* list (used here as temporary storage for useful prodrules).

VII. Delete all the clauses not associated with prodrules; that is, all the clauses that are not part of the input set.

VIII. WHILE (CHILLED list is not empty) DO:

- Browse the *CHILLED* list from head to tail and move into the *HEATED* list those prodrules having all the antecedents evaluated to be true.
- If it is a plausible reasoning session, search the *UNUSED* list for refractory prodrules (prodrules concluding on known facts), and move them into the *HEATED* list (if possible).

- Fire all prodrules from the *HEATED* list; that is, move them into the *FIRED* list.

G5.2. Forward Chaining Algorithm

WHILE (no stop rule was fired yet) DO the following three steps:

I. Browse the *UNUSED* list from head to tail and:

- Move into the HEATED list the prodrules having all the antecedents evaluated to be true.
- Move into the *CHILLED* list all the prodrules with an antecedent evaluated to be false.
- If it is not a plausible reasoning session, then move into the *CHILLED* list all the refractory prodrules (prodrules concluding on known facts).

II. If the *HEATED* list is empty, then move into the *HEATED* list the first default prodrule (if any) from the *UNUSED* list.

III. If *HEATED* list is empty, the return failure; otherwise, fire the rule placed at the head of *HEATED* list.

- If it is a plausible reasoning session, then fire all additional rules concluding on the same fact with the first one.

REFERENCES

SHINGHAL, R., (1992), *Formal Concepts in Artificial Intelligence*, Chapman and Hall, 1992.

WOS, L., OVERBEEK, R., LUSK, E., BOYLE, J., (1984) *Automated Reasoning: Introduction and applications*, Prentice-Hall, 1984.

APPENDIX H: CONSTANTS AND C++ CLASS TEMPLATES

H1. BUILDING ENVELOPE CONSTANTS

```
      // Building Occupancy Types (as defined by ASHRAE 90.1/1989)
#define BT_NONE          0  // Undefined
#define BT_ASSEMBLY      1  // Large unpartioned spaces (hall, church)
#define BT_OFFICE        2  // Office buildings, banks and alikes
#define BT_RETAIL        3  // Mercantile oriented buildings
#define BT_WAREHOUSE     4  // Storage facilities
#define BT_SCHOOL        5  // Educational buildings
#define BT_HOTEL         6  // Hotel & motel type buildings
#define BT_RESTAURANT    7  // Restaurant, Fast Food , Cafeteria, etc
#define BT_HEALTH        8  // Hospitals & other health care buildings
#define BT_RESIDENTIAL   9  // Residential buildings excepting low rise
#define BT_DWELLING     10  // Detached single or two family dwelling

      // Building Height Types (as defined by the number of floors)
#define LOW_RISE            0            // 3 floors or less
#define MEDIUM_LOW   (LOW_RISE + 1)      // 3 < floors <= 10
#define MEDIUM_HIGH  (MEDIUM_LOW + 1)    // 10 < floors <= 20
#define HIGH_RISE    (MEDIUM_HIGH + 1)   // over 20 floors

      // Building Size Types (as defined by the total floor area)
#define AREA_SMALL          0            // 5000 ft2 or less
#define AREA_10      (AREA_SMALL + 1)    // 10000 ft2 < area <= 10000 ft2
#define AREA_15      (AREA_10 + 1)       // 10000 ft2 < area <= 15000 ft2
#define AREA_20      (AREA_15 + 1)       // 15000 ft2 < area <= 20000 ft2
#define AREA_50      (AREA_20 + 1)       // 20000 ft2 < area <= 50000 ft2
#define AREA_75      (AREA_50 + 1)       // 50000 ft2 < area <= 75000 ft2
#define AREA_LARGE   (AREA_75 + 1)       // over 75000 ft2 of floor area

      // Envelope System Types
#define ST_NONE          0  // none
#define ST_WALL          1  // wall
#define ST_GLAZING       2  // glazing
#define ST_ROOF          3  // roof
#define ST_FLOOR         4  // floor

      // Envelope Layer Types
#define LT_NONE          0  // none
#define LT_FINISH        1  // interior finish
#define LT_COATING       2  // coating (sheathing)
#define LT_MEMBRANE      3  // roofing membrane
#define LT_CLADDING      4  // wall cladding
#define LT_PANEL         5  // exterior panel
#define LT_INSULATION    6  // insulation
#define LT_STRUCTURE     7  // structural member
#define LT_OTHER         8  // glazing & others
#define LT_CAVITY        9  // airspace

      // Insulation Positions (as used by System Performance Criteria)
#define IP_EXTERIOR      0
```

```
#define IP_INTEGRAL      1
#define IP_INTERIOR      2

      // Layer/System Face Position
#define INNER      TRUE  // inner side (closer to the indoor environ.)
#define OUTER      FALSE // outer side (closer to the outer environment)

      // Flow Directions
#define IN2OUT     TRUE  // indoor -> outdoor
#define OUT2IN     FALSE // outdoor -> indoor

      // Load Types
#define COOLING    FALSE
#define HEATING    TRUE

      // Types of HVAC Systems
#define CVHC       1  // Constant Volume Heating & Cooling System
#define CVR        2  // Const. Volume with Terminal Reheat System
#define CVDD       3  // Constant Volume Double Duct System
#define VAV        4  // Variable Air Volume System
#define FPFC       5  // Four Pipe Fan Coil System
#define WLHP       6  // Water Loop Heat Pump System
#define RCHP       7  // Reverse Cycle Heat Pump System

      // ASHRAE Schedule Types
#define OCCUPANCY  0  // occupancy schedule
#define LIGHTING   1  // lighting schedule
#define HVAC       2  // HVAC system schedule
#define SWH        3  // service hot water schedule

      // ASHRAE Day Types
#define WEEKDAY    0
#define SATURDAY   1
#define SUNDAY     2

      // Canadian Location Types (Provinces)
#define P_NONE     0  // None
#define P_ALTA     1  // Alberta
#define P_BC       2  // British Columbia
#define P_MAN      3  // Manitoba
#define P_NB       4  // New Brunswick
#define P_NFL      5  // New Foundland
#define P_NS       6  // Nova Scotia
#define P_NWT      7  // North Western Territory
#define P_ONT      8  // Ontario
#define P_QUE      9  // Quebec
#define P_PEI     10  // Prince Edward Island
#define P_SASK    11  // Saskatchewan
#define P_YT      12  // Yukon Territory

      // Location Data
#define LD_WT25    1  // Winter temperature 2.5%
#define LD_WT10    2  // Winter temperature 1.0%
#define LD_SDBT25  3  // Summer dry-bulb temperature 2.5%
#define LD_SWBT25  4  // Summer wet-bulb temperature 2.5%
```

```
#define LD_HDD18  5  // Heating degree-days above 18.3 degree C

      // Degree-Days & Degree-Hours Types
#define HDD18     0  // Degree-Days below 18 degrees C
#define HDD10     1  // Degree-Days below 10 degrees C
#define CDD18     2  // Degree-Days above 18 degrees C
#define CDD10     3  // Degree-Days above 10 degrees C
#define CDH27     4  // Degree-Hours above 27 degrees C

      // Annual Average Daily Incident Solar Energy Types
#define VSN       0  // on NORTH facade
#define VSEW      1  // on EAST/WEST facade
#define VSS       2  // on SOUTH facade

      // Facade Orientations
#define NORTH     0
#define EAST      1
#define SOUTH     2
#define WEST      3

      // Miscellaneous Constants
const double dZeroTemp = -273.15;       // Zero-absolute (degrees C)
const double dGravity = 9.80665;        // gravity constant (m/sec2)
const double PI = 3.14159;              // number PI
const double dTempError = 0.001;        // error for temp. calculations
const double dAirSpaceHeight = 0.3048; // 304.8 mm (1 ft)
const double dAirFilmHeight = 0.3048;  // 304.8 mm

      // Physical Signature Parameter Types
typedef enum
{
   SIGN_STRUCTURE,      // structural signature
   SIGN_MASS,           // mass signature (heat capacity)
   SIGN_THERMAL,        // thermal resistance signature
   SIGN_MOISTURE,       // vapour resistance signature
} SIGNATURE;

      // Matching Types for Cross-Section Analysis
typedef enum
{
   MT_NONE,             // No match
   MT_AIR,              // Air cavity
   MT_WOOD,             // Wood
   MT_METAL,            // Metal, Steel, Aluminium, etc
   MT_STONE,            // Concrete, Cement, Gypsum, Masonry, Mortar
   MT_OTHER,            // Composite
} MATCH_TYPE;
```

H2. C++ CLASS HIERARCHY

Table H-1: Class hierarchy in CRED

Base Class	Derived Class	Derived Class	Top Class	Description
CObject	*CfcBldgMaterial*		*CfcLayer*	Building Material Layer
CPtrList			*CfcSystem*	Envelope Subsystem
CObject	*CfcPrototypeBldg*		*CfcBuilding*	Building Envelope
CObList			*CfcBldgDB*	Building Envelope Database (Case Library)
CObject			*CfcSite*	Building Location Descriptor
CObject	*CfcLayerNN*	*CfcInputLayer*		Neural Network Layer
		CfcOutputLayer	*CfcMiddleLayer*	
CPtrList			*CfcNeuralNetwork*	Neural Network
CObject			*CfcFilter*	Neural Network Filter
CObject			*CfcMatcher*	Case Matcher
CObject		*CfcReasoningEngine*		Automatic Reasoning Engine

H3. BUILDING MATERIAL CLASS

CfcBldgMaterial is the encapsulation of the generic building material used for building envelope layers.

```
class CfcBldgMaterial : public CObject
{
protected:
    CString sMatName;      // material's name
    CString sPattern;      // material's bitmap

    double dThickness;     // thickness of material (m)
    double dDensity;       // density (kg/m3)
    double dThermalRes;    // thermal resistance (r-value) (m2C/W)
    double dVapourRes;     // water vapour resistance (m2secPa/ng)
    double dSpecificHeat;// specific heat (kJ/kgC)
    double dCost;          // material cost ($/m2)
    double dEmIn;          // inner side emisivity (0 ... 1)
    double dEmOut;         // outer side emisivity (0 ... 1)

public:
       // Constructors/Destructor
    CfcBldgMaterial(const char* pszName = NULL);
    CfcBldgMaterial(const CfcBldgMaterial& BldgMat);
    virtual ~CfcBldgMaterial(void);

       // Overridables
    virtual void Serialize(CArchive& ar);

       // Operations
    const char* GetName(void);
    void GetName(CString& sName);
```

```
    void SetName(const char* pszName);
    void SetName(CString& sName);

    const char* GetPattern(void);
    void GetPattern(CString& sName);
    void SetPattern(const char* pszName);
    void SetPattern(CString& sName);

       // Physical properties
    double GetThickness(void);
    void SetThickness(double dValue);
    double GetDensity(void);
    void SetDensity(double dValue);
    double GetThermalRes(void);
    void SetThermalRes(double dValue);
    double GetVapourRes(void);
    void SetVapourRes(double dValue);
    double GetSpecificHeat(void);
    void SetSpecificHeat(double dValue);
    double GetHeatCapacity(void);           // heat capacity (kJ/m2C)
    double GetCost(void);
    void SetCost(double dValue);
    double GetEmissivity(BOOL bSide);
    void SetEmissivity(double dValue);
    void SetEmissivity(double dValue, BOOL bSide);

       // Cloning
    void Copy(const CfcBldgMaterial& BldgMat);

private:
       CfcBldgMaterial& operator = (const CfcBldgMaterial&);
};
```

H4. ENVELOPE LAYER CLASS

CfcLayer is derived from the building material class (i.e. CfcBldgMaterial) and encodes the generic behavior of building envelope layer.

```
class CfcLayer : public CfcBldgMaterial
{
protected:
    int nLayerType;        // layer type
    double dTin;           // inner surface dry-bulb temperature (deg.C)
    double dTout;          // outer surface dry-bulb temperature (deg.C)
    double dRHin;          // inner surface relative humidity (0 ... 1)
    double dRHout;         // outer surface relative humidity (0 ... 1)
    double dHeatFlow;      // heat flow In->Out (W/m2)
    double dVapourFlow;    // water vapour flow In->Out (ng/m2sec)
    double dCondense;      // concealed condensation (ng/m2sec)

public:
       // Constructor/Destructor
    CfcLayer(int nType, const char* pszName = NULL);
```

```
    CfcLayer(int nType, const CfcBldgMaterial& BldgMat);
    CfcLayer(const CfcLayer& Layer);
    virtual ~CfcLayer(void);

       // Overridables
    virtual void Serialize(CArchive& ar);

       // Operations
    int GetLayerType(void);
    void SetLayerType(int nType);
    BOOL GetTypeName(CString& sName);

       // Dynamic properties
    double GetTemperature(BOOL bSide);
    void SetTemperature(double dValue, BOOL bSide);
    double GetRH(BOOL bSide);
    void SetRH(double dValue, BOOL bSide);
    double GetHeatFlow(void);
    double GetVapourFlow(void);
    double GetDP(BOOL bSide);
    double GetCondense(void);
    BOOL HasCondense(void);

       // Calculations
    void HeatFlow(double dHeatFlowIn);
    BOOL VapourFlow(double dFlowIn, double dFlowOut, double& drh);
    void AirCavityFlow(double dHeatFlowIn, double dTiltAngle);

       // Cloning
    void Copy(const CfcLayer& Layer);

private:
    CfcLayer& operator =(const CfcLayer&);
};
```

H5. ENVELOPE SUBSYSTEM CLASS

CfcSystem is an ordered list of layers (class CfcLayer) counted from inside toward outside. Class CfcSystem is used to encode envelope subsystems such as *Wall*, *Roof*, and *Glazing*.

```
class CfcSystem : public CPtrList
{
protected:
    CString sSystemName; // system name
    int nSystemType;     // system type
    double dTiltAngle;   // tilt angle with horizontal (degrees)
    double dRfin, dRfout;// r-value for inside/outside air films
    BOOL bSCondense;     // TRUE for surface condensation

       // Design environment
    double dTin;         // inner surface dry-bulb temperature (deg.C)
    double dTout;        // outer surface dry-bulb temperature (deg.C)
    double dRHin;        // inner surface relative humidity (0 ... 1)
```

```
   double dRHout;       // outer surface relative humidity (0 ... 1)

public:
      // Constructor/Destructor
   CfcSystem(int nType = ST_NONE, const char* pszName = NULL);
   CfcSystem(const CfcSystem& System);
   virtual ~CfcSystem(void);

      // Overridables
   virtual void Serialize(CArchive& ar);
   virtual void RemoveAll(void);

      // Operations
   void Init(void);
   int GetSystemType(void);
   void SetSystemType(int nType);
   BOOL GetTypeName(CString& sName);

   const char* GetName(void);
   void GetName(CString& sName);
   void SetName(const char* pszName);
   void SetName(CString& sName);

   double GetTiltAngle(void);
   void SetTiltAngle(double dValue);
   double GetTemperature(BOOL bSide);
   void SetTemperature(double dValue, BOOL bSide);
   double GetSurfaceTemperature(BOOL bSide);
   double GetSurfaceHumidity(BOOL bSide);
   double GetRH(BOOL bSide);
   void SetRH(double dValue, BOOL bSide);
   void SetDesignEnvironment(double dtin, double drhin,
                             double dtout, double drhout);
      // Layer manipulation
   int GetLayerIndex(int nPos, BOOL bSide);
   BOOL AddLayer(CfcLayer* pLayer, BOOL bSide);
   BOOL InsertLayer(CfcLayer* pLayer, int nPos, BOOL bSide);
   BOOL SwapLayers(int nPos1, int nPos2, BOOL bSide);
   BOOL MoveLayer(int nOldPos, int nNewPos, BOOL bSide);
   BOOL RemoveLayer(int nPos, BOOL bSide);
   BOOL RemoveLayer(CfcLayer* pLayer, BOOL bDelete = TRUE);
   CfcLayer* GetLayerByPosition(int nPos, BOOL bSide);
   CfcLayer* GetLayerByType(int nType, BOOL bSide);

      // Overall physical properties
   double GetAirFilmResistance(BOOL bSide);
   double GetAirFilmResistance(double dThermalRes, BOOL bSide);
   double GetThickness(void);
   double GetThermalRes(BOOL bAirFilms = TRUE);
   double GetVapourRes(void);
   double GetVapourRes(int nStartIndex, int nEndIndex);
   double GetRequiredVapourRes(void);
   double GetHeatCapacity(void);
   double GetCost(void);
   double GetEmissivity(BOOL bSide);
```

```
      // Dynamic properties
   double GetHeatFlow(void);
   double GetVapourFlow(void);
   double GetConcealedCond(void);
   BOOL HasConcealedCond(void);
   BOOL HasSurfaceCond(void);
   BOOL HasCondense(void);

      // Flow Calculations
   void HeatFlow(void);
   void VapourFlow(void);
   void Flow(void);

      // Matching functions
   CSize GetSignature(SIGNATURE nType);
   MATCH_TYPE GetMatchingType(int nLayerType);
   BOOL HasCavity(void);

      // Cloning
   void Copy(const CfcSystem& System);
   void Transform(CfcSystem& System);

protected:
   int CheckVapourFlow(double dFlow, int nStartIndex, int nEndIndex);

private:
   CfcSystem& operator =(const CfcSystem&);
};
```

H6. ASHRAE PROTOTYPE BUILDING CLASS

```
class CfcPrototypeBldg : public CObject
{
protected:
   int nBuildingType;    // building type (BT_...)
   double dFloorArea;    // total floor area (m2)
   double DD[5];         // degree-days & degree hours
   double VS[3];         // annual avg. daily solar energy on facade
   double DR;            // average temperature range for warmest month

public:
      // Constructors/Destructor
   CfcPrototypeBldg(int nType = BT_NONE, double dTotalArea = 0.0);
   CfcPrototypeBldg(const CfcPrototypeBldg& Building);
   virtual ~CfcPrototypeBldg(void);

      // Overridables
   virtual void Serialize(CArchive& ar);

      // Operations
   virtual void Init(void);
   int GetBuildingType(void);
```

```
    void SetBuildingType(int nType);
    double GetFloorArea(void);
    void SetFloorArea(double dValue);
    void GetClimaticEnvironment(double dd[5], double vs[3], double& dr);
    void SetClimaticEnvironment(double dd[5], double vs[3], double dr);
    void SetClimaticData(CfcSite& Site);

       // ASHRAE System Performace Criteria
    static double GetDefaultAspectRatio(BOOL bPrototype);
    static double GetDefaultHC(void);
    double GetWallArea(double dAspectRatio = (double)0);
    double GetWindowToWallRatio(double dInternalLoads = (double)0);
    double GetShadingCoeff(void);
    double GetInternalLoads(void);
    double GetUvalue(int nSystemType);

       // Building Loads
    double GetHeatingLoad(double dW2WR, double dAspectRatio,
                          double dUwall, double dUglazing, double dHCin,
                          double dShaddingCoeff, double dProjFactor,
                          double dInternalLoads,
                          int nInsulationPosition = IP_INTEGRAL);
    double GetCoolingLoad(double dW2WR, double dAspectRatio,
                          double dUwall, double dUglazing, double dHCin,
                          double dShaddingCoeff, double dProjFactor,
                          double dInternalLoads,
                          int nInsulationPosition = IP_INTEGRAL);
    double GetBuildingLoad(double dW2WR, double dAspectRatio,
                           double dUwall, double dUglazing,
                           double dHCin,double dShaddingCoeff,
                           double dProjFactor, double dInternalLoads,
                           int nInsulationPosition = IP_INTEGRAL);
       // System Performance Criteria
    double GetCriteria(BOOL bCriteria,
    int nInsulationPosition = IP_INTEGRAL);
    double GetCoolingCriteria(int nInsulationPosition = IP_INTEGRAL);
    double GetHeatingCriteria(int nInsulationPosition = IP_INTEGRAL);
    double GetPrototypeLoad(int nInsulationPosition = IP_INTEGRAL);
    BOOL SystemPerformanceCriteria(double dW2WR, double dAspectRatio,
                                   double dUwall, double dUglazing,
                                   double dHC, double dShaddingCoeff,
                                   double dProjFactor,
                                   double dInternalLoads,
                                   int nInsulationPosition);
       // Cloning Operations
    void Copy(const CfcPrototypeBldg& Building);

private:
    CfcPrototypeBldg& operator =(const CfcPrototypeBldg&);
};
```

H7. BUILDING ENVELOPE CLASS

```
class CfcBuilding : public CfcPrototypeBldg
{
protected:
   CString sBldgName;    // building name

   CfcSystem Roof;       // ROOF System
   CfcSystem Wall;       // WALL System
   CfcSystem Glazing;    // GLAZING System

   int nBldgFloors;      // number of floors
   double dBldgW2WR;     // Window-to-Wall ratio
   double dBldgAR;       // aspect ratio
   double dBldgSC;       // Shadding coefficient
   double dBldgPF;       // Projection factor
   double dBldgIL;       // Internal loads

      // Design environment
   double dTin;          // inner surface dry-bulb temperature (deg.C)
   double dTout;         // outer surface dry-bulb temperature (deg.C)
   double dRHin;         // inner surface relative humidity (0 ... 1)
   double dRHout;        // outer surface relative humidity (0 ... 1)

   double dMatchScore;   // Matching score

public:
      // Constructors/Destructor
   CfcBuilding(int nType = BT_NONE, double dTotalFloorArea = (double)0,
               int nFloors = 0, const char* pszName = NULL);
   CfcBuilding(const CfcBuilding& Building);
   virtual ~CfcBuilding(void);

      // Overridables
   virtual void Serialize(CArchive& ar);

      // Operations
   int GetHeightType(void);
   int GetSizeType(void);
   BOOL IsEmpty(void);

   virtual void Init(void);
   CfcSystem* GetRoof(void;
   CfcSystem* GetWall(void);
   CfcSystem* GetGlazing(void);
   static const char* GetTypeName(void);

   const char* GetName(void);
   void GetName(CString& sName);
   void SetName(const char* pszName);
   void SetName(CString& sName);

   int GetNumFloors(void);
   void SetNumFloors(int nFloors);
```

```
    double GetW2WR(void);
    void SetW2WR(double dValue);
    double GetAR(void);
    void SetAR(double dValue);
    double GetSC(void);
    void SetSC(double dValue);
    double GetPF(void);
    void SetPF(double dValue);
    double GetIL(void);
    void SetIL(double dValue);

    double GetTemperature(BOOL bSide);
    void SetTemperature(double dValue, BOOL bSide);
    double GetRH(BOOL bSide);
    void SetRH(double dValue, BOOL bSide);
    void SetDesignEnvironment(double dtin, double drhin,
                              double dtout, double drhout);
    void SetClimaticData(CfcSite& Site, BOOL bSetDefault = FALSE);

       // Flow Calculations
    void Flow(void);
    BOOL HasRoofCondense(void);
    BOOL HasWallCondense(void);
    BOOL HasCondense(void);

       // ASHRAE Criteria
    BOOL CheckRoof(void);
    BOOL CheckWall(int nInsulationPosition = IP_INTEGRAL);
    BOOL CheckCriteria(int nInsulationPosition = IP_INTEGRAL);
    BOOL CheckBuilding(BOOL bFull);
    double GetLoadType(BOOL bCriteria,
                       int nInsulationPos = IP_INTEGRAL);
    double GetLoad(int nInsulationPos = IP_INTEGRAL);

       // Matching Operations
    double GetMatchScore(void);
    void SetMatchScore(double dScore);

       // Cloning Operations
    void Copy(const CfcBuilding& Building);
    void Transform(CfcBuilding& Building);

private:
    CfcBuilding& operator =(const CfcBuilding&);
};
```

H8. BUILDING ENVELOPE CASE LIBRARY (DATABASE)

```
class CfcBldgDB : public CObList
{
protected:
    CString sDBFile;          // full path to the storage file
    int nSelIndex;            // index of selected building
```

```
public:
      // Constructor/Destructor
   CfcBldgDB(void);
   CfcBldgDB(const CfcBldgDB& BldgList);
   virtual ~CfcBldgDB(void);

      // Overridables
   virtual void RemoveAll(void);
   virtual void Serialize(CArchive& ar);

      // Operations
   const char* GetDBFile(void);
   void GetDBFile(CString& sName);
   void SetDBFile(const char* pszName);
   void SetDBFile(CString& sName);
   int GetSelIndex(void);
   void SetSelIndex(int nIndex);
   int GetIndex(CfcBuilding* pBldg = NULL);

      // Load/Save operations
   BOOL LoadDB(const char* pDBFile = NULL);
   BOOL SaveDB(const char* pDBFile = NULL);

      // Operations
   CfcBuilding* GetBuilding(int nPos = -1);
   BOOL AddBuilding(CfcBuilding* pBldg = NULL, BOOL bTail = TRUE);
   BOOL AddBuilding(const CfcBuilding& Building);
   BOOL InsertBuilding(CfcBuilding* pBldg, int nPos);
   BOOL RemoveBuilding(int nPos);
   BOOL RemoveBuilding(CfcBuilding* pBldg);

      // Matching Operations
   BOOL Sort(BOOL bAscending = FALSE);
   void SetupBldgName(void);

      // Cloning Operations
   void Copy(const CfcBldgDB& BldgList);

protected:
   BOOL Order(POSITION nPos1, POSITION nPos2);

private:
   CfcBldgDB& operator =(const CfcBldgDB&);
};
```

H9. BUILDING SITE (LOCATION) CLASS

```
class CfcSite : public CObject
{
public:
      // Location
   int nProvince;       // province code
```

```
    CString sProvince;  // name of selected province
    int nCity;          // city index within province list
    CString sCity;      // name of the location

       // Temperatures (degrees C)
    double dTin;        // design indoor temperature
    double dTout;       // design outdoor temperature
    double dWT25;       // winter temperature 2.5%
    double dWT10;       // winter temperature 1.0%
    double dSDBT25;     // summer dry-bulb temperature 2.5%
    double dSWBT25;     // summer wet-bulb temperature 2.5%

       // Relative humdities (%)
    double dRHin;       // design indoor relative humidity
    double dRHout;      // design outdoor relative humidity

       // ASHRAE-related data
    double dHDD18;      // heating degree-days below 18.3 deg.C
    double dHDD10;      // heating degree-days below 10 deg.C
    double dCDD18;      // cooling degree-days above 18.3 deg.C
    double dCDD10;      // cooling degree-days above 10 deg.C
    double dCDH27;      // cooling degree-hours below 26.67 deg.C
    double dVSN;        // avg. daily solar energy on North facade
    double dVSEW;       // avg. daily solar energy on East & West
    double dVSS;        // avg. daily solar energy on South facade
    double dDR;         // avg. daily range for the warmest month

public:
    CfcSite(void);
    CfcSite(const CfcSite& Site);
    virtual ~CfcSite(void);

       // General overrides
    virtual void Serialize(CArchive& ar);

       // Operators
    const CfcSite& operator=(const CfcSite& Site);

       // Helper functions
    static const char* GetProvinceName(int nProvinceCode);
    static CStringList* GetProvinceList(int nProvinceCode);
    static BOOL GetLocationData(int nProvinceCode,
                                const char* pszCityName,
                                int nDataType, double& dData);
       // Operations
    const char* GetProvince(void);
    const char* GetCity(void);
    void SetProvince(int nProvinceCode);
    void SetCity(int nCityIndex);
    void SetCity(const char* pszCityName);
    void SetLocation(int nProvinceCode, int nCityIndex);
    void SetLocation(int nProvinceCode, const char* pszCityName);
    CStringList* GetLocationList(void);
    void CheckLocationData(void);
```

```
protected:
   void Init(BOOL bFull = FALSE);
   void UpdateLocationData(void);
};
```

H10. ARTIFICIAL NEURAL NETWORK CLASSES

```
      // Artificial Neural Network Layer Types
typedef enum
{
   NN_NONE,              // unknown type
   NN_INPUT,             // input layer
   NN_MIDDLE,            // middle layer
   NN_OUTPUT             // output layer
} NN_LAYERTYPE;


class CfcLayerNN : public CObject
{
protected:
   int nLayerType;          // layer type
   int nNumInputs;          // number of inputs for this layer
   int nCells;              // number of outputs for this layer
   double* pCells;          // Array of outputs for this layer
   CfcLayerNN* pInLayer;    // input layer (if any)

public:
   CfcLayerNN(int nInputs = 0, int nOutputs = 0);
   virtual ~CfcLayerNN(void);

      // Overridables
   virtual void Serialize(CArchive& ar);

      // Operations
   int GetLayerType(void);
   BOOL IsValid(void);
   void SetInputLayer(CfcLayerNN* pLayer);
   int GetLayerSize(void);
   double GetCellValue(int nIndex);
   virtual void Init(void);

      // Functional methods
   virtual BOOL CalculateOutput(void) = 0;

      // Tracing functions
   void TraceOutputs(void);
   virtual void TraceCell(int nIndex);

protected:
   virtual BOOL SetupLayer(int nInputs, int nOutputs);
   virtual void Cleanup(void);
   friend class CfcNeuralNetwork;
```

```
private:
    CfcLayerNN(const CfcLayerNN&);
    CfcLayerNN& operator =(const CfcLayerNN&);
};


class CfcInputLayer : public CfcLayerNN
{
protected:
    double dNoiseFactor;     // noise factor
    double* pInputVector;    // original inputs

public:
    CfcInputLayer(int nInputs = 0, int nOutputs = 0);
    virtual ~CfcInputLayer(void);

       // Operations
    void SetNoiseFactor(double dValue);
    virtual void Init(void);

       // Functional methods
    virtual BOOL CalculateOutput(void);

protected:
    virtual BOOL SetupLayer(int nInputs, int nOutputs);
    virtual void Cleanup(void);
    friend class CfcNeuralNetwork;

private:
    CfcInputLayer(const CfcInputLayer&);
    CfcInputLayer& operator =(const CfcInputLayer&);
};


class CfcOutputLayer : public CfcLayerNN
{
protected:
    double* pWeights;           // layer's weights
    double* pOutputVector;      // expected output to a given input
    double* pOutputErrors;      // array of errors at output
    double* pBackErrors;        // array of errors back-propagated
    double* pCumDeltas;         // for momentum calculations
    double* pPastDeltas;        // for momentum calculations

public:
    CfcOutputLayer(int nInputs = 0, int nOutputs = 0);
    virtual ~CfcOutputLayer(void);

       // Overridables
    virtual void Serialize(CArchive& ar);
       // Operations
    virtual void Init(void);

       // Functional methods
    virtual BOOL CalculateOutput(void);
```

```
    virtual BOOL CalculateError(double& dError);
    BOOL RandomizeWeights(void);
    BOOL CaseUpdating(const double dBeta, const double dAlpha,
                      BOOL bUpdateWeights);
    BOOL EpochUpdating(BOOL bUpdateWeights);

       // Tracing functions
    void TraceWeights(int nIndex);
    void TraceErrors(void);
    virtual void TraceCell(int nIndex);

protected:
    virtual BOOL SetupLayer(int nInputs, int nOutputs);
    virtual void Cleanup(void);
    friend class CfcNeuralNetwork;

private:
    CfcOutputLayer(const CfcOutputLayer&);
    CfcOutputLayer& operator =(const CfcOutputLayer&);
};


class CfcMiddleLayer : public CfcOutputLayer
{
public:
    CfcMiddleLayer(int nInputs = 0, int nOutputs = 0);
    virtual ~CfcMiddleLayer(void);

private:
    CfcMiddleLayer(const CfcMiddleLayer&);
    CfcMiddleLayer& operator =(const CfcMiddleLayer&);
};


class CfcNeuralNetwork : public CPtrList
{
protected:
    BOOL bEnabled;          // TRUE when network is operational

public:
    CString sNetworkFile;   // Storage file for this neural network

public:
    CfcNeuralNetwork(void);
    CfcNeuralNetwork(const CfcNeuralNetwork& srcNeuralNet);
    virtual ~CfcNeuralNetwork(void);

       // Overridables
    virtual void Serialize(CArchive& ar);
    virtual void RemoveAll(void);

       // Operations
    BOOL IsEnabled(void);
    const char* GetNetworkFile(void);
    void GetNetworkFile(CString& sFileName);
```

```
    void SetNetworkFile(const char* pszFileName);
    void SetNetworkFile(CString& sFileName);
    void Init(void);
    void Copy(const CfcNeuralNetwork& srcNeuralNet);

       // Layers manipulation
    CfcLayerNN* GetLayer(int nIndex);
    CfcInputLayer* GetInputLayer(void);
    CfcOutputLayer* GetOutputLayer(void);
    BOOL AddLayer(int nOutputs = 0);
    int GetInputSize(void);
    int GetOutputSize(void);
    void SetNoiseFactor(double dValue);
    void SetInputVector(double* pInputVector);
    void SetOutputVector(double* pOutputVector);

       // I/O method
    BOOL SaveToFile(const char* pFileName = NULL);
    BOOL LoadFromFile(const char* pFileName = NULL);
    BOOL MemoryTransfer(CMemFile& MemFile, BOOL bLoad);

       // Functional methods
    BOOL TrainingPath(double* pInput, double* pOutput, double& dError);
    BOOL TestingPath(double* pInput, double* pOutput);
    BOOL ForwardProp(void);
    BOOL BackwardProp(double& dError);
    BOOL RandomizeWeights(void);
    BOOL CaseUpdating(const double dBeta, const double dAlpha,
                      BOOL bUpdateWeights);
    BOOL EpochUpdating(BOOL bUpdateWeights);

       // Tracing functions
    void GetInfo(CString& sInfo, BOOL bFull = FALSE);
    void OnTraceNetwork(BOOL bFull = FALSE);
    void OnTraceData(void);

protected:
    BOOL CheckNetwork(void);

private:
       CfcNeuralNetwork& operator =(const CfcNeuralNetwork&);
};
```

H11. NEURAL NETWORK FILTER CLASS

```
class CfcFilter : public CObject
{
protected:
    CfcNeuralNetwork NeuralNet;// neural network filter
    CString sFilterFile;       // storage file
    CString sTrainInFile;      // input set file for training
    CString sTrainOutFile;     // output vectors file for training
    CString sTestInFile;       // input set file for testing
```

```
    CString sTestOutFile;       // output vectors file for testing

public:
    int nFilterCount;           // maximum count of selected buildings

            // Building parameters: Type, Height, Size, W2W Ratio, AR
    BOOL bBldgParams[5];

        // Wall & Roof Signatures
    BOOL bWallParams[8];
    BOOL bRoofParams[8];

        // Training parameters
    double dBeta;               // learning rate
    double dAlpha;              // momentum
    double dNF;                 // noise factor
    double dThresholdError;     // threshold error (to stop training)
    int nMaxCycle;              // maximum number of training cycle
    BOOL bRandomize;            // TRUE to randomize the weights
    BOOL bCaseUpdating;         // TRUE/FALSE for case/epoch updating

public:
    CfcFilter(void);
    CfcFilter(const CfcFilter& srcFilter);
    virtual ~CfcFilter(void);

        // General overrides
    virtual void Serialize(CArchive& ar);
    const CfcFilter& operator =(const CfcFilter& srcFilter);
        // Filtering operations
    BOOL IsEnabled(void)            {return (NeuralNet.IsEnabled());}
    void DoPrepareTraining(CfcSite& Site, CfcBldgDB& BldgList);
    void DoPrepareTesting(CfcSite& Site, CfcBldgDB& BldgList);
    BOOL DoTrain(void);
    BOOL DoTest(CfcBuilding* pBuilding, CfcSite& Site,
                CfcBldgDB& BldgListIn, CfcBldgDB& BldgListOut);

        // Load/Save operations
    void Init(void);
    BOOL Load(const char* pszFileName = NULL);
    BOOL Save(const char* pszFileName = NULL);

        // Editing Neural Network filter
    void OnClearNN(void);
    void OnInitNN(void);
    void OnAddLayerNN(int nCells;
    void OnGetInfoNN(CString& sInfo, BOOL bFull = FALSE);
    void OnCopyNN(const CfcFilter& srcFilter);

protected:
    void SetupFiles(void);
    int DoPrepareInVector(CfcBuilding* pBldg, double* dVector);
    int DoPrepareOutVector(CfcBuilding* pBldg, double* dVector);
    void Copy(const CfcFilter& srcFilter);
};
```

H12. CASE MATCHER CLASS

```
class CfcMatcher : public CObject
{
protected:
   CString sWeightsFile;      // storage file

public:
      // Building matching (Type, Floors, Floor Area, W2W Ratio, AR)
   double dBuildingWeights[5];

      // Layer type matching
   double dGlazingWeights[10];
   double dWallWeights[10];
   double dRoofWeights[10];

      // Building material type matching
   double dGlazingWeightsEx[10];
   double dWallWeightsEx[10];
   double dRoofWeightsEx[10];

public:
   CfcMatcher(void);
   CfcMatcher(const CfcMatcher& srcMatcher);
   virtual ~CfcMatcher(void);

      // General overrides
   virtual void Serialize(CArchive& ar);
   const CfcMatcher& operator =(const CfcMatcher& srcMatcher);

      // Matching operations
   BOOL DoMatch(CfcBuilding* pRefBldg, CfcBldgDB& BldgList);
   double GetMatchingScore(CfcBuilding* pBldg1, CfcBuilding* pBldg2);
   double GetMatchingScore(CfcSystem* pSystem1, CfcSystem* pSystem2);

      // Load/Save operations
   void Init(void);
   BOOL Load(const char* pszFileName = NULL);
   BOOL Save(const char* pszFileName = NULL);

protected:
   void Copy(const CfcMatcher& srcMatcher);
};
```

H13. AUTOMATIC REASONING MODULE (ARM) CONSTANTS

```
      // Options flags
#define E_CGROUND     0x01  // reduce clause ground
#define E_GROUND      0x02  // reduce global ground
#define E_DEMOD       0x04  // demodulation
#define E_PARAMOD     0x08  // paramodulation
```

```
#define E_HYPER       0x10  // hyper-resolution
#define E_HRLTD       0x20  // constrained hyper-resolution

      // Features of clauses & literals
#define CL_NONE       0x00
#define CL_EQUAL      0x01  // equal predicate within clause
#define CL_NON_EQUAL  0x02  // negated equal predicate within clause
#define CL_GROUND     0x04  // literals are grounded
#define CL_FGROUND    0x08  // literals are fully grounded
#define CL_DEAD       0x10  // clause is useless
#define CL_USED       0x20  // clause was used to answer a question
#define CL_CHASTE     0x40  // chaste clause flag (chaste literal)

      // Features of rules
#define R_DEFAULT     0x01  // default rule
#define R_STOP        0x02  // stop rule (if fired, inference stops)
#define R_LINK        0x04  // inferenced fact is linked to more than
                            // one prodrule consequent
      // Execution mode
#define EXPERTISE_MODELLING   FALSE     // Expertise modelling
                                          (expert system)
#define RESOLUTION_REFUTATION TRUE      // Resolution refutation

      // Size Constants
#define MAX_ARGS        5  // max. number of arguments within literals
#define MAX_LITERALS   10  // max. number of literals within clauses
#define MAX_CLAUSES  1000  // maximum preset number of clauses
#define MAX_LISTS       4  // number of lists for clauses
#define MAX_STATUS      4  // number of status alternatives

      // Global typedefs for ARE
typedef enum
{     // CLAUSE source
   CS_NONE            =  0,       // clause of unknown origin
   CS_PREMISSE        =  1,       // premisse <= input set
   CS_QUESTION        =  2,       // question to be answered (input set)
   CS_GOAL            =  3,       // goal <= input set
   CS_INSTANCE        =  4,       // specialization
   CS_RESOLVENT       =  5,       // resolution
   CS_HYPER_RESOLVENT=  6,       // hyper-resolution
   CS_ANSWER          =  7,       // answer to question
   CS_DEMODULANT      =  8,       // demodulation
   CS_PARAMODULANT    =  9,       // paramodulation
   CS_FACT            = 10,       // produced through inference
} CLAUSE_SOURCE;

typedef enum
{     // CLAUSE type
   CT_MIXED           =  0,       // mixed clause
   CT_POSITIVE        =  1,       // positive clause
   CT_NEGATIVE        =  2,       // negative clause
} CLAUSE_TYPE;

typedef enum
{     // SET-of-SUPPORT employed
```

```
    SOS_NONE            =  0,
    SOS_WEIGHT_SUP      =  1,       // clause hierarchy based on weight
    SOS_GOAL_SUP        =  2,       // support is derived from the goal
    SOS_POS_SUP         =  3,       // support provided by pos. clauses
    SOS_NEG_SUP         =  4,       // support provided by neg. clauses
} SET_OF_SUPPORT;

typedef enum
{     // Types of lists for packing clauses
    PL_DEPTH            =  0,       // identifies depth-oriented list
    PL_LENGTH           =  1,       // identifies length-oriented list
    PL_SET_OF_SUPPORT   =  2,       // identifies set-of-support list
    PL_CONSISTENT       =  3,       // list of clauses not in set-of-support
} PACK_LIST;

typedef enum
{     // TERM type
    TT_NONE             =  0,       // unknown type
    TT_CONSTANT         =  1,       // constant
    TT_VARIABLE         =  2,       // variable
    TT_FUNCTION         =  3,       // function
    TT_PREDICATE        =  4,       // predicate
} TERM_TYPE;

typedef enum
{     // RULE status
    RS_UNUSED           =  0,       // not yet browsed
    RS_CHILLED          =  1,       // READY/CHILLED in BACK/FWD chaining
    RS_HEATED           =  2,       // ready to be fired
    RS_FIRED            =  3,       // rule was fired
} RULE_STATUS;

typedef enum
{
    FWD_CHAINING,                   // Forward Chaining
    BACK_CHAINING,                  // Backward Chaining
} EM_STRATEGY;

typedef enum
{
    UNIT_RESOLUTION,      // Unit Resolution
    FEWEST_LITERALS,      // Fewest Literals Preference
    BREADTH_FIRST,        // Depth Saturation
    LBS,                  // Linear Back-Chaining with Selector on
                          // Assertions
    SLD,                  // Linear Resolution with Selector on Definite
                          //Clauses
} RR_STRATEGY;

typedef struct SYMBOL
{
    TERM_TYPE nType;                // symbol's type
    char* pszName;                  // symbol's name
    struct SYMBOL* pPrev;           // previous link
    struct SYMBOL* pNext;           // next link
```

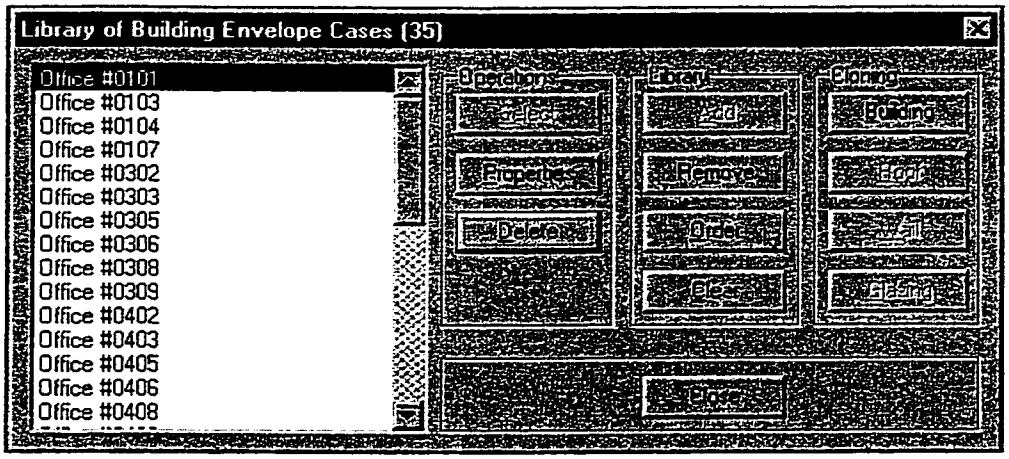
```
} SYMBOL;

typedef struct TERM
{
   TERM_TYPE nType;                  // term's type
   UINT nFeature;                    // additional features for the term
   SYMBOL* pSymbol;                  // term's symbol
   UINT nArg;                        // no. of arguments for complex terms
   struct TERM* pArgs[MAX_ARGS];     // argument list for complex terms
} TERM;

typedef struct SUBSTITUTION
{
   TERM* pTermIn;                    // term to be introduced
   TERM* pTermOut;                   // term to be replaced
   struct SUBSTITUTION* pPrev;       // previous link
   struct SUBSTITUTION* pNext;       // next link
} SUBSTITUTION;

typedef struct CLAUSE
{
   CLAUSE_SOURCE nOrigin;            // clause origin
   CLAUSE_TYPE nType;                // clause type
   UINT nFeature;                    // features for the clause
   UINT nID;                         // clause id (generation index)
   UINT pID[MAX_LITERALS + 1];       // id for clause's parents
   UINT nDepth;                      // depth level of the clause
   UINT nDeep;                       // maximum depth of clause's terms
   UINT nLength;                     // number of literals within clause
   UINT nCnum;                       // number of remaining consorts
   UINT nCindex;                     // current consort index
   float fWeight;                    // weight/certainty factor
   BOOL bUsed;                       // TRUE if clause part of proof tree
   BOOL bSign[MAX_LITERALS];         // list of literals signs
   TERM* pLiteral[MAX_LITERALS];     // list of literals within clause
   SUBSTITUTION* pSub;               // substitutions (if any)
   struct CLAUSE* pPrev[MAX_LISTS];  // previous link in clauses list
   struct CLAUSE* pNext[MAX_LISTS];  // next link in clauses list
} CLAUSE;

typedef struct RULE
{
   RULE_STATUS nStatus;              // rule's status
   UINT nFeature;                    // rule's features
   CLAUSE* pClause;                  // Horn clauses in the rule
   CLAUSE* pFact;                    // new fact produced by "firing"
   UINT nID;                         // ID number (generation order)
   float fRuleProb;                  // probability (certainty) of the rule
   float fFactProb;                  // contribution to fact's probability
   struct RULE* pPrev[MAX_STATUS];   // previous link in rules lists
   struct RULE* pNext[MAX_STATUS];   // next link in rules lists
} RULE;
```

H14. AUTOMATIC REASONING MODULE (ARM) CLASSES

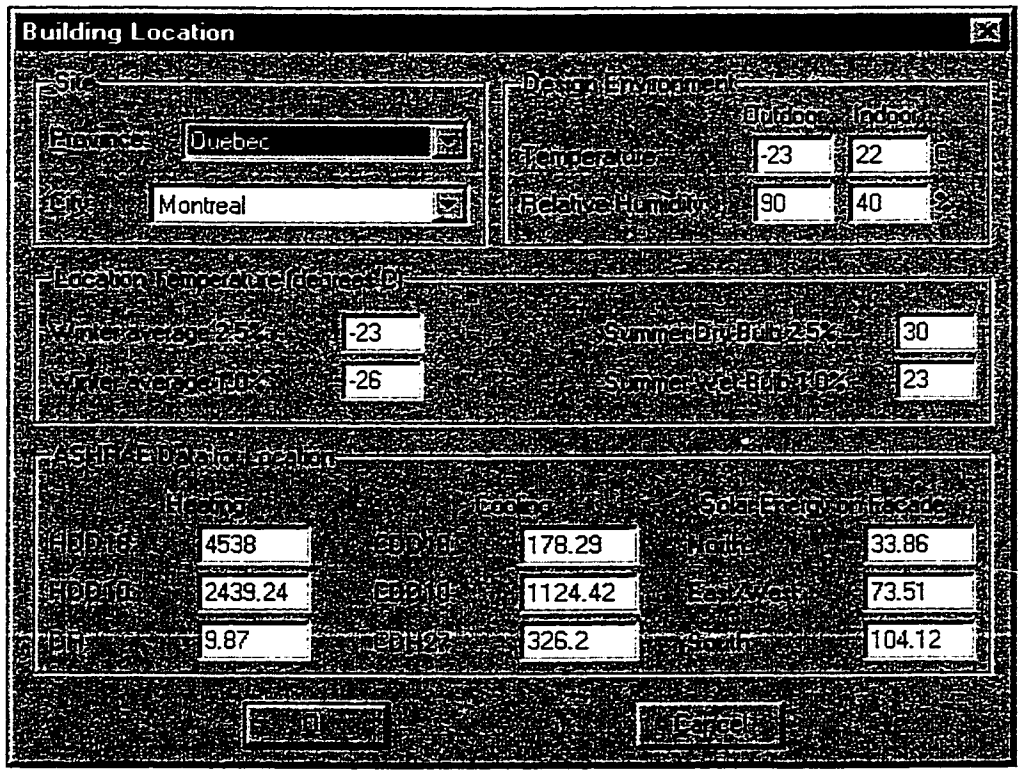
```
class CfcReasoningEngine : public CObject
{
public:
   BOOL bExecutionMode;     // Expertise Modelling/Resolution Refutation
   UINT nStrategy;          // Strategy type
   UINT nSupport;           // Set-of-support
   UINT nOptions;           // Options flags

   CLAUSE ClauseList;       // Clauses list
   RULE RuleList;           // Rules list

protected:
   BOOL bReasoningMode;     // Plausible Reasoning: certainty factor < 1
   SYMBOL ConstList;        // Constants list
   SYMBOL VarList;          // Variables list
   SYMBOL FuncList;         // Functions list
   SYMBOL PredList;         // Predicates list
   int nResolventCount;     // number of generated resolvents

public:
   CfcReasoningEngine(void);
   virtual ~CfcReasoningEngine(void);

      // Overridables
   virtual void Serialize(CArchive& ar);
      // Operations
   CLAUSE* GetFacts(BOOL bAll);

      // Expertise Modelling Strategies
   BOOL ForwardChaining(void);
   BOOL BackwardChaining(BOOL bAllAnswers);  // Uses LBS

      // Resolution Refutation Strategies
   BOOL BreadthFirst(UINT nSetSupport);
   BOOL FewestLiterals(UINT nSetSupport);
   BOOL WeightingStrategy(void);
   BOOL UnitResolution(void);
   BOOL OnePassUnitResolution(void);
   BOOL ModifiedUnitResolution(void);
   BOOL SetOfSupportResolution(void);
   BOOL StrategyLBS(BOOLEAN bAllAnswers);
   BOOL StrategySLD(void);

protected:
   void Init(void);
   void Cleanup(void);

private:
   CfcReasoningEngine(const CfcReasoningEngine&);
   CfcReasoningEngine& operator =(const CfcReasoningEngine&);
};
```

APPENDIX J: SAMPLE TRACE OF CRED EXECUTION STEPS

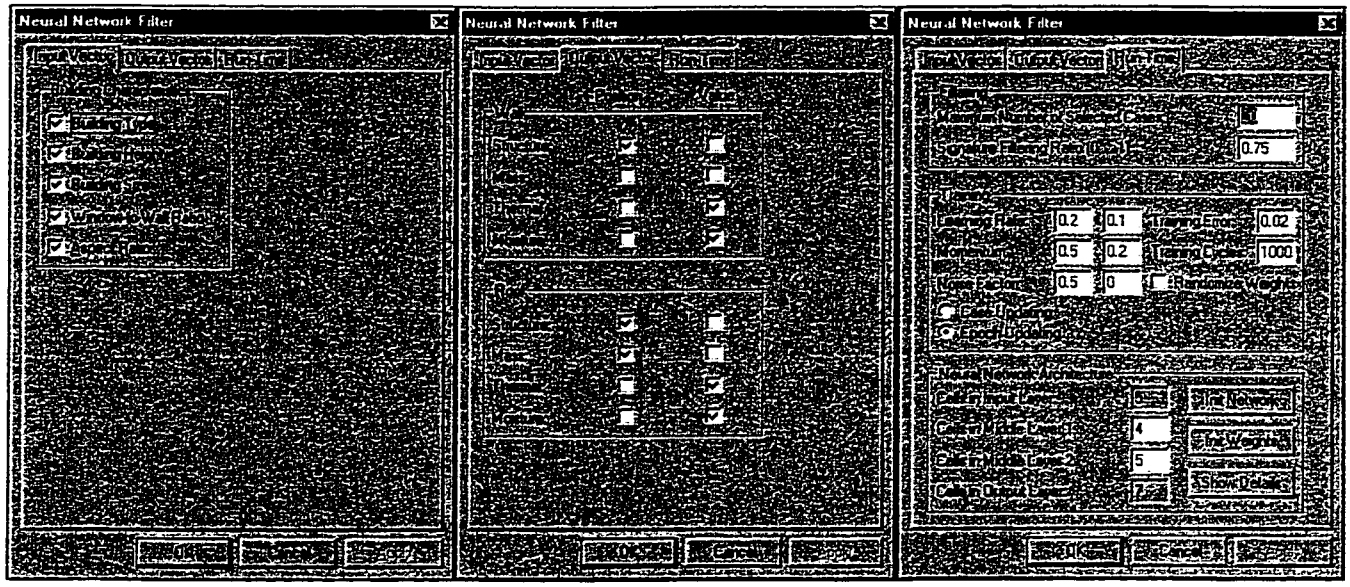
J1. CASE LIBRARY



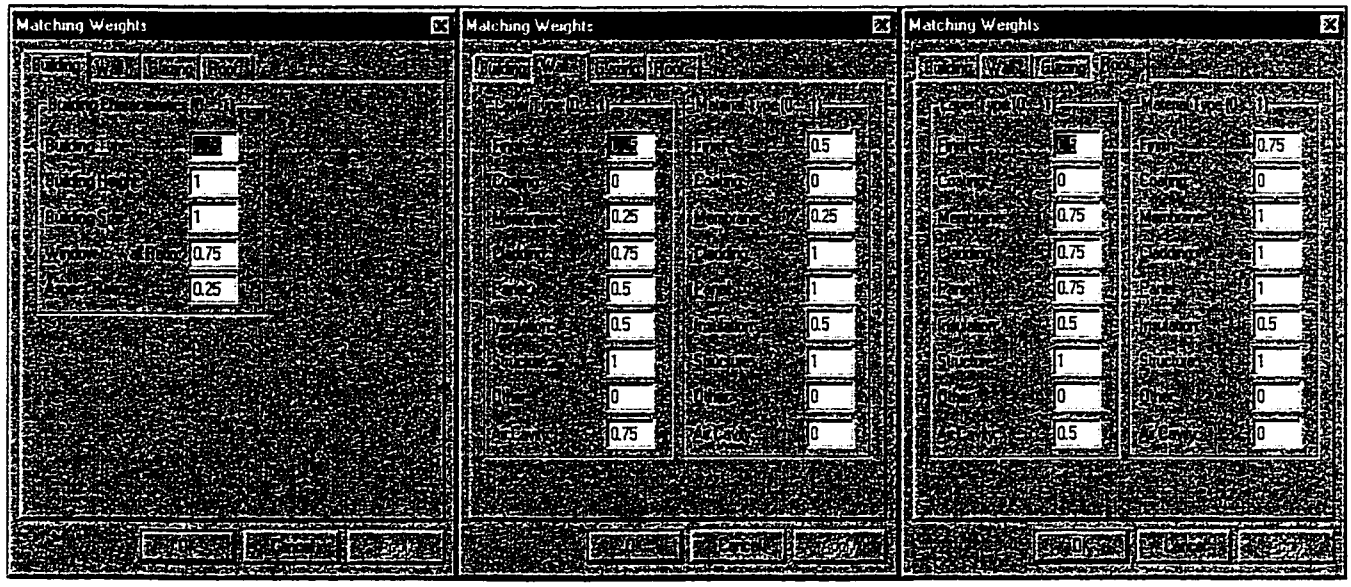
J2. BUILDING LOCATION



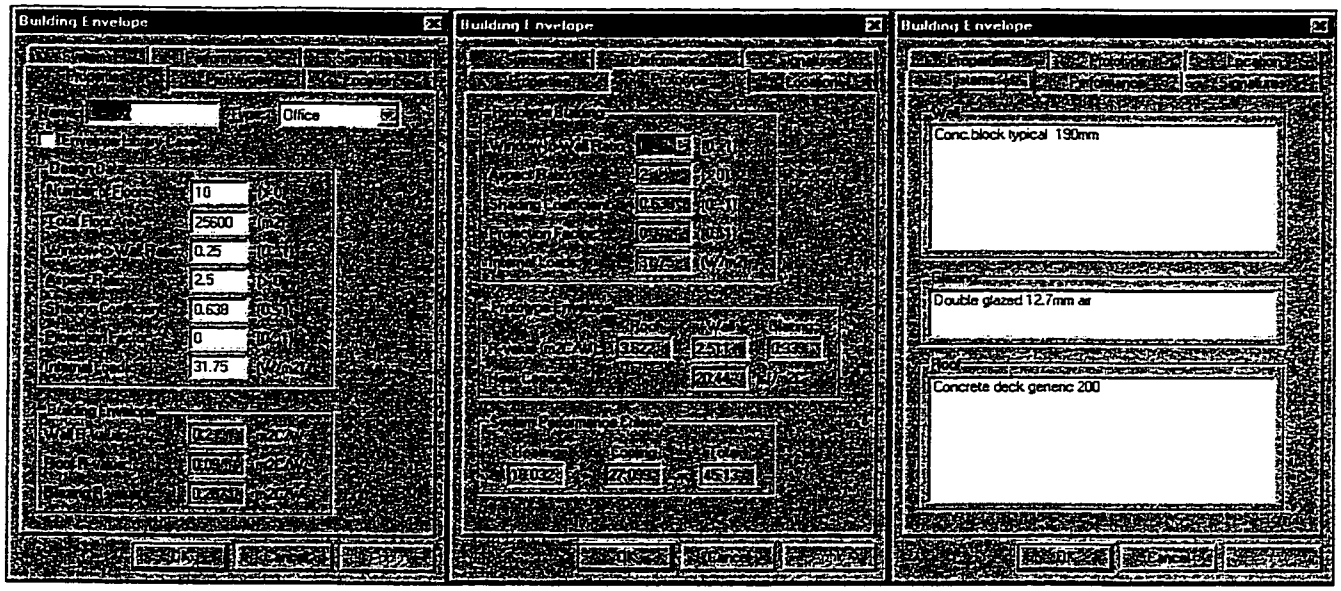
J3. ANN-BASED FILTER SETUP



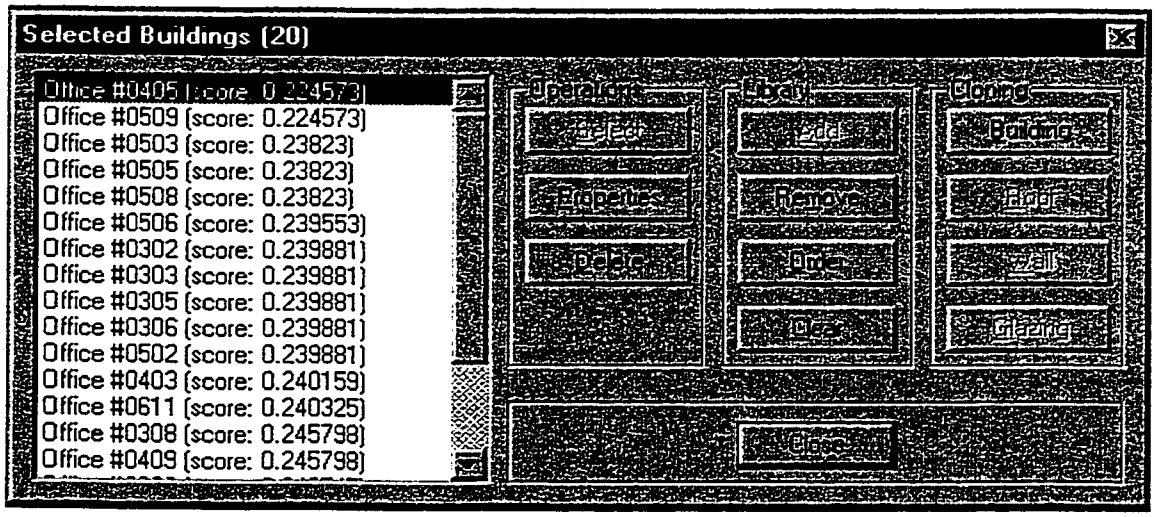
J4. MATCHING WEIGHTS



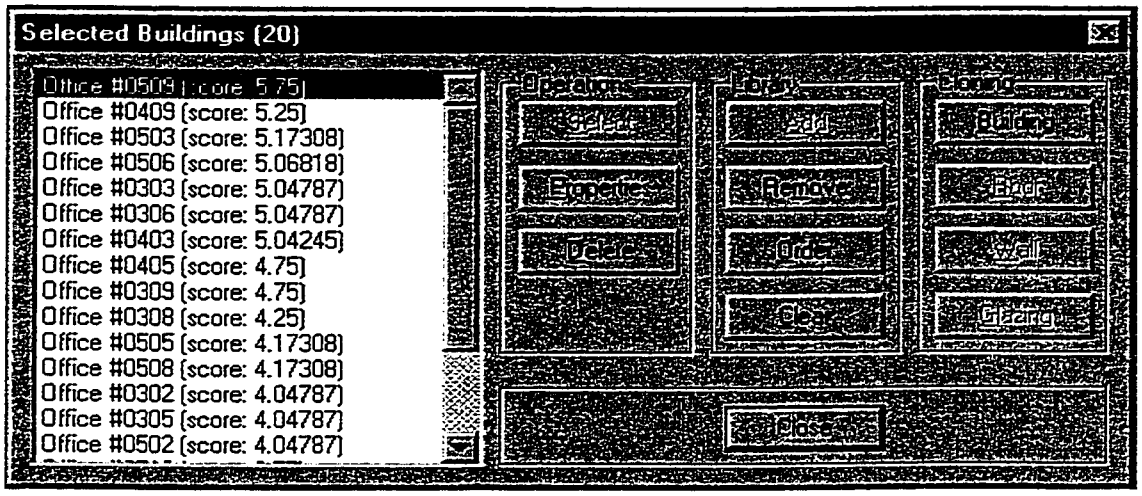
J5. DESIGN CASE



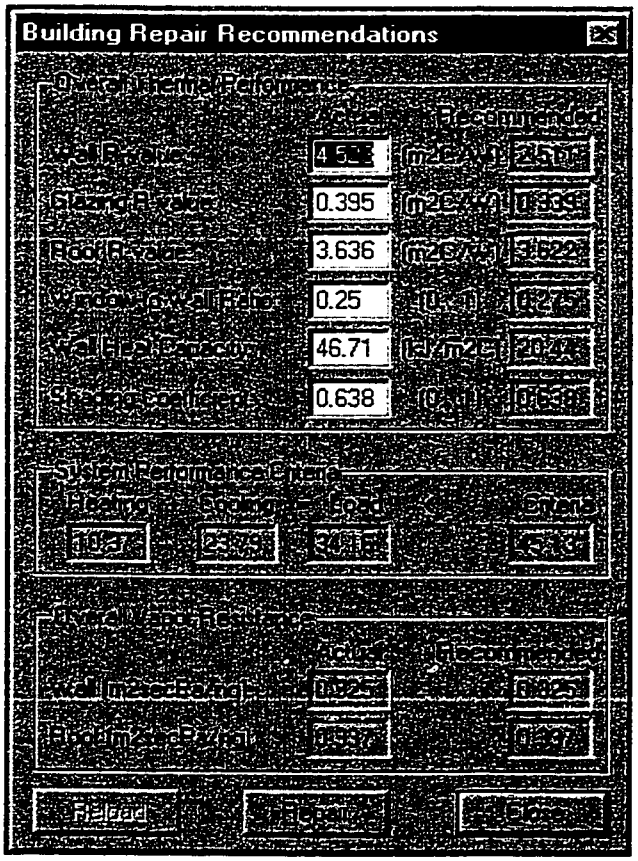
J6. PRELIMINARY SELECTION AND PHYSICAL-SIGNATURE FILTERING



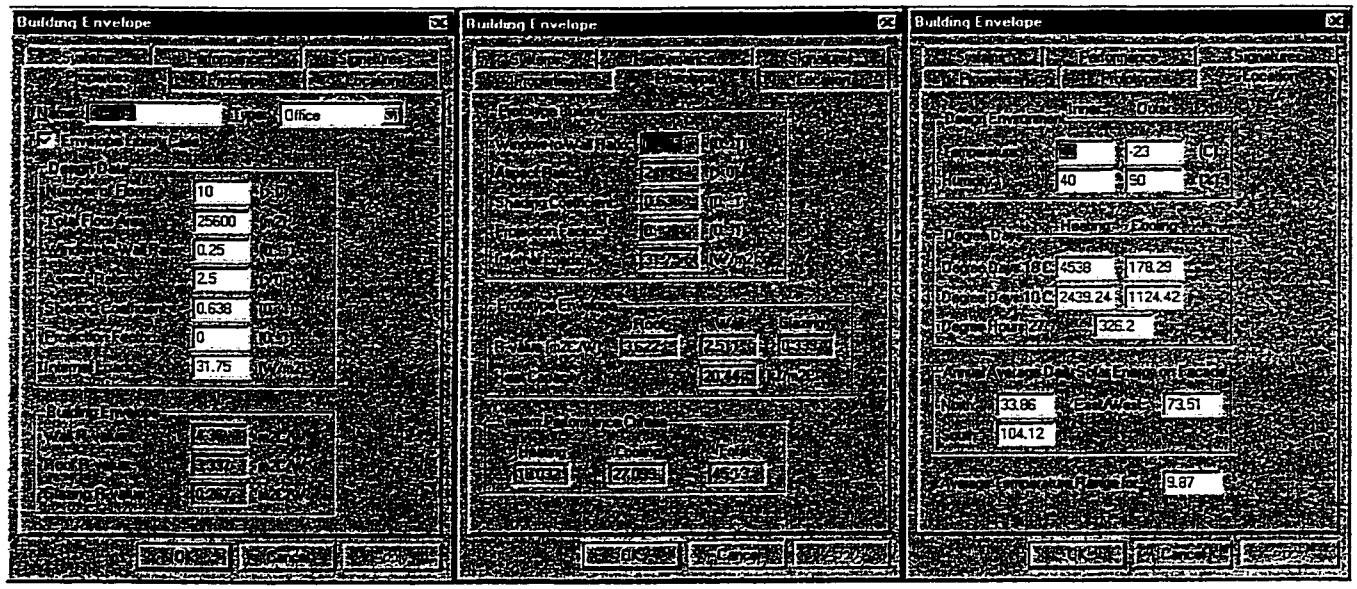
J7. FULL MATCHING

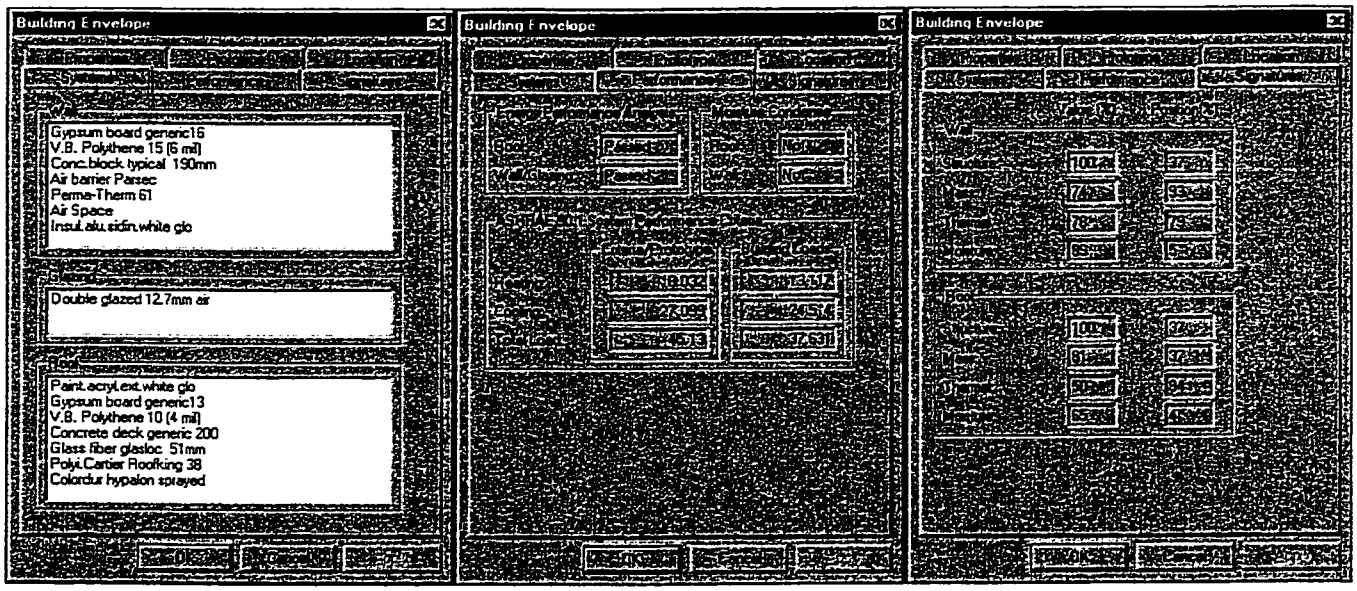


J8. REPAIR



J9. RECOMMENDED ENVELOPE





GLOSSARY OF TERMS AND ABBREVIATIONS

Access	A step in the CBR flow when the available case library is browsed for best-matching cases.
Activation Function	Mathematical relationship defining how the input flow is passed through an ANN node (or cell).
Adaptation	CBR phase during which the retrieved cased is modified to meet the input requirements. Also known as Transformation or Mapping.
Adapted Case	Outcome of the Adaptation (or Transformation) phase. It is a case that preserves the characteristics and usefulness of the retrieved library case while conforming to most (if not all) design requirements. Also known as "modified" case.
Adjacency Validation	Mechanism to validate a new building envelope by matching adjacent layers within the cross-section.
AI	Artificial Intelligence (computer science field concerned with solving problems through means that attempt to simulate human thinking).
Anchor Layer	Wall/Roof layer having the most impact on the physical signature associated with a given parameter.
ANN	Artificial Neural Networks (a computerized model for neural connectivity found in the human brain).
ARM	Automated Reasoning Module. The rule-based engine of CRED.
Aspect Ratio	The ratio of building width over the building depth.
ASHRAE	American Society of Heating, Refrigerating, and Air-Conditioning Engineers.
Automated Reasoning	Process of generating a reasoning outcome by eliciting new facts through firing rules based on a predefined control strategy.
Back Propagation	ANN learning method (Rumelhart et at, 1986) that

	computes an error at each layer of a feed-forward network and propagates this error backwards to determine appropriate improvements to the weights.
Base Case	Library case selected for adaptation. It also known as "source" case. If selection is based on matching, then the base case is usually the best-match case.
BESA	Building Energy Systems Analysis (Software program)
Best-Match Case	Library case deemed to be the closest match for a given set of input requirements.
Breadth-First Search	Search strategy that examines all the rules or objects on the same level of the hierarchy before examining any rules or objects on the next lower level.
Building Class	A set of buildings sharing the same building vector.
Building Vector	The set of building attributes (such as building type, height, size, aspect ratio, and window-to-wall ratio) identified as primary features for a building-envelope case.
Building Type	ASHRAE-based building classification according to the preferred occupancy type.
Building Height Type	Building classification according to the number of floors.
Building Size Type	ASHARE-based building classification according to the total floor area (i.e. ranges used for selecting the HVAC system).
CAD	Computer-Aided Design
CBR	Case-Based Reasoning (AI paradigm where old situations, called cases, are recalled to find solutions to new problems)
Case	A contextualized piece of knowledge embedding a domain specific lesson about how to achieve a given goal. It usually denotes a specific problem situation including its interpretation, solution, and possible annotations.
Case Library (Memory)	Collection of cases related to a given domain (such as building-envelope design).

CDD	Cooling Degree-Days
CDH	Cooling Degree-Hours
Class Signature	The average physical signature for the set of buildings sharing the same building vector.
Clause	Disjunction of literals
CMHC	Canada Mortgage and Housing Corporation
COM	Component Object Model evolved from OLE paradigm (Microsoft architecture for distributing computing where components expose binary interfaces for dynamic binding at run-time).
Comparison Function	Function used to assess the degree of similarity between two entities (i.e. cases). See as well Evaluation Function.
Compliance Checking	Set of methods used to verify if a given envelope case complies with the design requirements as recommended by the ASHRAE Standards and by local construction lows.
Cooling Criteria	System Performance equation for testing if the average cooling load of a given building is below the recommended value (i.e. building prototype load) for the location.
CRED	Case Reasoner for Envelope Design
DB	Database (Collection of data records structured according to a predefined model and organized in an orderly fashion).
Demodulation	Automated reasoning technique used in resolution refutation to resolve clauses and to demodulate from one clause into another clause. It can be found as either backward or forward demodulation.
Depth-First Search	Search strategy in which rules or objects on the highest level of a search branch are examined prior to moving to the next search branch (immediately below the rule or object currently examined).
Design Case	Incomplete case initially assembled based on design requirements and further refined to define the Target Case.

Design Context	The set of design requirements including building location, building type, and building geometry.
Design Signature	The physical signature associated with the design case. It is actually the class signature corresponding to the design vector.
Design Vector	The building vector associated with the Design Case.
Digitized Feature	Case feature having a numerical value that is encoded as an enumeration (that is, as belonging to a one group from a limited set of possible value groups).
DOE-2	Building energy performance simulation software created by the U.S. Department of Energy and used by ASHARE to infer the System Performance Criteria.
DR	Daily range average for temperatures of warmest month.
Envelope Component (Subsystem)	One of the major parts of the building envelope such as Wall, Roof, or Glazing.
Envelope Layer	Building material component to be found in the cross-section of a building-envelope subsystem.
Epoch Updating	ANN training method that updates the cell weights only after all the data samples in the training set are examined once by the network.
Euclidean Distance	Method to assess the degree of similarity of two patterns based on the geometric distance between the individual features exhibited by those patterns.
Evaluation Function	Function used to assess the total matching distance between the target case and a library case. It is a refined version of the Comparison Function.
Exemplar (Case) Updating	ANN training method that updates the cell weights each time a data sample from the training set is examined by the network.
Experiential Knowledge	Knowledge gained from hands-on experience, usually known as *surface knowledge* (in contrast to the *deep knowledge* of formal principal and theories).

Filtering	Processes of narrowing down the number of cases to consider for matching through an ANN driven selection anchored on the design signature.
HDD	Heating Degree-Days
Heating Criteria	System Performance equation for testing if the average heating load of a given building is below the recommended value (i.e. building prototype load) for the location.
Heuristic Rule	Rule capturing the heuristics an expert uses to solve a problem.
HVAC System	Heating, Ventilation and Air Conditioning System
Inconsistency	Formula that is false under all of its interpretations (as opposed to a tautology).
Index	Feature of a case linked to the context in which that case teaches a unique lesson.
Indexing	Operation of identifying those salient features of the case that are further used as primary retrieval criteria.
Inference	Process by which new facts are derived from established facts.
Input Set	Initial set of input parameters for a reasoning process, or the input data to an ANN.
KB	Knowledge Base (collection of information and rules encapsulating the facts and heuristics about a domain and used throughout a reasoning process).
KBES	Knowledge-Based Expert Systems (AI software programs solving problems through a KB inference process).
Knowledge Acquisition	Process of locating, collecting, and refining knowledge.
Knowledge Representation	Method used to encode facts and relationships in a KB.
Learning	Dynamic process through which an AI software is updating its Knowledge Base to enhance the quality of its reasoning outcome.

Learning Rate	ANN training parameter (a.k.a. Beta), usually constant with a value between 0 and 1, that affects the updating rate applied to cell weights.
Learning Rule	Algorithm used by an ANN during the training phase to update the weights associated with each network cell.
Lexical Parsing	Decomposition operation applied to an input text to identify and isolate "words" and "structures" corresponding to a pre-defined grammar.
Literal	Atom or a negation of an atom.
Mapping	See Adaptation
Matching	Process of identifying within a case library the case that best fits a given set of input requirements.
Matching Threshold	Maximum numbers of cases to be retained for/after the matching process.
Message Passing	OOP method in which an object sends a "message" to another object by triggering a procedure stored within it.
MFC	Microsoft Foundation Classes (collection of generic C++ classes for the Win16 and Win32 environments).
Modified Case	See Adapted Case.
MOP	Memory Organization Packet (see Schank, 1982)
Nearest-Neighbor Algorithm	Pattern recognition method to find the closest entity to a given target by computing the distance between that target and all the entities in its neighborhood.
Object	Conceptual entity that has a collection of attributes and methods that might be either private or exposed to the outside world.
OOP	Object-Oriented Programming
Paradigm Case	Case encapsulating a significant and unique piece of information that makes it worth remembering (and a prime candidate for the case library).

Paramodulation	Automated reasoning technique used in resolution refutation to resolve clauses and to paramodulate from one clause into another clause. It can be found as either backward or forward demodulation.
Pattern Recognition	AI paradigm concerned with identifying and matching data patterns within a raw stream of information.
Prototype Building	Collection of building-envelope parameters recommended by the System Performance Criteria as either target values when aiming for energy efficiency, or as average values to be used as replacemenst when faced with missing data.
Performance Simulation	A computerized process modeling the behavior of something in the real world to retrieve a theoretical estimation of its performance.
Performance Tests	Collection of test procedures included in the Performance Simulation.
Physical Signature	Heuristic parameter attempted to encode the cross-section description of an envelope system along a predefined physical parameter.
Prodrule	Production rule (and defined as a set of antecedents paired with a consequent).
Propagation Rule	ANN rule defining how outputs from network layer cells are combined into the overall input of a cell from the next network layer.
Proposition	Automated reasoning declaration that can have either a true or a false value.
Ranking	Sorting operation based on the score achieved during a matching step.
Recommended Case	Successful outcome of a CBR system (a valid library case modified to match a different set of input requirements).
Repair	CBR step attempting to correct the failures detected within a Modified Case.
Repaired Case	Case successfully "repaired" during a CBR Repair phase.

Resolution Refutation	Procedure by which resolution proves theorems.
Resolvent	Clause deduced by resolving two "parent" clauses.
Retrieval	Process of accessing the case library to retrieve suitable cases, usually for matching purposes.
Rule-Based System	Program representing knowledge by means of rules.
R-value	Thermal resistance of a building-envelope element (layer or subsystem).
Salient Feature	Primary feature of a case usually employed to classify the case, or to index the case in the library.
Similarity Metrics	Set of comparison strategies and functions used to assess the degree of similarity between two cases.
Solution Case	See Recommended Case.
Source Case	See Target Case.
SPC	System Performance Criteria
SDA	Suitable Design Alternative
Signature Vector	Set of physical signature values assembled for a given case.
Target Case	The initial case (usually incompletely described) defined as the goal to be matched throughout the CBR process.
Tautology	Formula true under all of its interpretation (as opposed to an inconsistency).
TOP	Thematic Organization Packet (or Unit). See Schank, 1982
Training Phase (ANN)	Process of adjusting the weights of an ANN by feeding the network with input data that should be matched by known output vectors.
Transformation	See Adaptation
Transfer Mechanism	Collection of rules and procedures used to modify a case to fit a different set of input requirements.

Term	Constant, variable, function, or predicate
Testing Phase (ANN)	Process of feeding a trained ANN with input data in order to retrieve the corresponding output vectors.
Unmatched Feature	Case feature identified as not matching the input requirements.
Updating	Process of dynamically refining the case library following reasoning processes that yielded significant outcomes.
U-value	Thermal conductance of an envelope element (the inverse of R-value).
Validation Tests	Collection of procedures used to check if a new case conforms to the commonly accepted domain knowledge.
Verified Case	Case passing the validation and performance tests.
VSEW	Annual average daily solar energy on East & West facades.
VSN	Annual average daily solar energy on North facade.
VSS	Annual average daily solar energy on South facade.
Weak Theory Domain	A subject matter area or a problem solving task that cannot be fully formalized through a set of knowledge rules and/or mathematical relationships.
Weighting Coefficient	Constant numerical value (usually between 0 and 1) defining the weight associated with a given parameters within the Evaluation Function.
Window-to-Wall Ratio	The percentage of glazing area of the total wall area.